.NET BlogBook

4. Ausgabe

Norbert Eder
Kai Gloth



Dipl. Ing. (FH) Norbert EDER
http://Weblog.norberteder.com

Norbert Eder ist als Software Architect und Technologieberater tätig. Er beschäftigt sich seit den Anfängen mit dem Microsoft .NET Framework. Zusätzlich engagiert er sich für die .NET Community und versucht sein Wissen an Hilfesuchende weiter zu geben.

Sie können ihn unter me@norberteder.com erreichen.

Community Projekte

http://www.dotnet-blog.com
http://www.dotnetcasts.com



Kai Gloth
http://Weblog.veloursnebel.de

Kai Gloth arbeitet als Softwareentwickler und studiert Informatik. Er beschäftigt sich hauptsächlich mit ASP.NET und versucht die Begeisterung für diese Technologie an Einsteiger weiterzugeben.

Sie können ihn unter kai@veloursnebel.de erreichen.

Community Projekte

http://www.dotnetcasts.com

Unterstützt durch:

Das Magazin für die Microsoft-Community
visual studio one



Gewinnchance:

In dieser Ausgabe befindet sich ein Gewinncode. Senden Sie diesen an csharp@gmx.at bis spätestens 30.11.2007 und gewinnen Sie wahlweise ein Jahresabonnement der visual studio one oder der ASP.NET Professional. Bitte geben Sie in Ihrem Email folgende Daten an:

- Name
- Adresse
- Email-Adresse
- Wie wurden Sie auf das .NET BlogBook aufmerksam?

FÜNF Gewinner werden unter allen Teilnehmern am 1.12.2007 gezogen und öffentlich per Namen bekannt gegeben. Der Gewinner ist damit einverstanden, dass die relevanten Daten für die Gewinnübermittlung an den ppedv Verlag weitergegeben werden. Darüber hinaus erfolgt keine Weitergabe der Daten. Der Rechtsweg ist ausgeschlossen. Die Teilnahme von Mitarbeitern der ppedv AG, sowie .NET Casts am Gewinnspiel ist ausgeschlossen.

1..NET BlogBook

1.1.Vorwort

Die vierte Ausgabe des .NET BlogBooks ist im Gegensatz zur dritten Ausgabe wieder mit jeder Menge neuer Einträgen gefüllt. Zusätzlich wurde wieder ein Großteil des Inhalts sowohl auf Fehler hin überprüft, als auch inhaltlich verbessert. Wer das .NET BlogBook bis dato nicht kennt: Es handelt sich hierbei um eine Sammlung und Überarbeitung von Weblog-Beiträgen der in dieser Ausgabe genannten Autoren. Im Laufe der Zeit sammelten sich viele hilfreiche Artikel an, die nur online zur Verfügung stehen. In vielen Fällen stellt sich jedoch eine Offline-Variante als sehr hilfreich und sinnvoll heraus. Hauptsächlich bei Programmierarbeiten beim Kunden, der aufgrund Sicherheitsrichtlinien keinen Zugriff in das Internet erlaubt, oder während einer Zugfahrt mit vergessener UMTS-Karte, oder aber auch einfach nur zum Ausdrucken und Durchstöbern.

Derzeit umfasst dieses BlogBook die Einträge von zwei Weblogs zu unterschiedlichen .NET Themen. Um die Inhalte aktuell zu halten, werden diese einmal im Quartal erweitert und überarbeitet um so neue Inhalte zu schaffen, als auch bestehende Inhalte optimaler aufzubereiten.

Nachfolgend finden Sie eine Übersicht der Erscheinungstermine an welchen aktualisierte Ausgaben veröffentlicht werden.

- 15. Januar
- 15. April
- 15. Juli
- 15. Oktober

Neue Versionen können unter http://www.dotnetcasts.com bezogen werden und stehen kostenlos zur Verfügung.

1.2.Ziel des Projektes

Wie die meisten Projekte wurde auch für das .NET BlogBook ein Ziel definiert. Unterschiedliche .NET Inhalte (von grundlegenden Beispielen zum .NET Core Framework, über designtechnische Grundlagen bis hin zur Webentwicklung) sollen einfach aufbereitet und übersichtlicher Form vermittelt werden. Das Zielpublikum wird sowohl durch den .NET-Einsteiger, als auch durch den Profi geprägt.

Ebenfalls wird ein weiterer Ausbau dieses Projektes ins Auge gefasst. Zusätzlich zum BlogBook werden auch Livecasts sowie Podcasts veranstaltet.

Kurzdefinition: Ein umfassendes How-To-Werk für alle .NET Entwickler und solche, die es noch werden wollen.

1.3.Was wird in diesem Buch nicht abgedeckt?

Kein Ziel ist es, absolute Grundlagen-Themen zu vermitteln, die in einschlägigen Büchern in den ersten Kapiteln zu finden sind. Die ersten Kapitel in diesem Buch beschäftigen sich dementsprechend bereits mit hilfreichen Tipps und Ratschlägen. Die typischen Einführungskapitel eines Programmier-Buches gibt es nicht und wird es auch in naher Zukunft nicht geben. Im Vordergrund sollen hilfreiche Tipps stehen, als auch jede Menge Code-Beispiele, um dem Entwickler eine schnelle und einfache Unterstützung zu bieten. Zusätzlich wird zu jedem Code-Beispiel entsprechendes Hintergrundwissen vermittelt.

Dieses Buch ist auch nicht geeignet, von A bis Z durchgelesen zu werden. Vielmehr werden Tipps und Tricks zu bestimmten Gebieten angeboten. Es ist als Nachschlagewerk konzipiert und wird dies auch zukünftig bleiben.

1.4.Verbesserungsvorschläge

Wie jedes andere Projekt auch, lebt dieses nicht nur vom Aufwand der Verantwortlichen, sondern auch von den Konsumenten des Ergebnisses. In diesem Sinne sind wir für alle Verbesserungsvorschläge dankbar. Ebenfalls nehmen wir jegliche Unterstützung gerne an, sofern sie diesem Projekt dienlich ist.

1.5.Rechtlicher Hinweis

Sämtliche Inhalte des .NET BlogBooks entstammen den Federn von Kai Gloth und Norbert Eder. Dementsprechend dürfen die Inhalte nicht ohne Zustimmung kopiert, vervielfältigt oder anderweitig verwendet werden. Eine Nichteinhaltung zieht rechtliche Konsequenzen nach sich.

2..NET Framework

Dieses Kapitel behandelt das .NET Framework. Dies bedeutet hier sind zahlreiche Hilfen, Ratschläge und Diskussionen zu sämtlichen Framework-Themen aufgelistet. Es finden sich Informationen zu Windows Forms, zum Basis-Framework, zu ASP.NET und vielem mehr.

2.1.Allgemein

2.1.1.Klassen- und Namespace-Informationen erfolgreich finden

Informationen zu Klassen können auf unterschiedlichste Weise gefunden werden. Die grundlegenden Möglichkeiten betreffen

- Object Browser
- Microsoft Developer Network

Der **Object Browser** ist äußerst hilfreich, wenn beispielsweise nach Klassen gesucht wird, jedoch der Namespace nicht bekannt ist. Alle Klassen, die in bereits referenzierten Assemblies vorhanden sind, können so gefunden werde. Dies betrifft alle Klassen der .NET Framework Core. Zusätzlich können Methoden-, Eigenschafts- und Vererbungs-Informationen bezogen werden.

Microsoft steckt sehr viel Aufwand und Mühe in das **Microsoft Developer Network** (MSDN). Darin finden sich unter zahlreichen Artikeln und Hilfestellungen auch die Dokumentationen zu den einzelnen .NET Framework Versionen. Jede Klasse des .NET Frameworks ist darin aufgelistet und großteils mit Beispielen versehen. So lassen sich die entsprechenden Namespaces finden, als auch Hinweise wie die Klassen verwendet werden, ob sie threadsicher sind und viele weitere Informationen. Ein muss für jeden .NET Entwickler.

Weitere Ressourcen
Zusätzlich finden sich eine Menge weiterer Ressourcen zum Thema .NET im Internet. Wer Beispiele sucht ist auf CodeProject gut aufgehoben. Wer ständig aktuelle Informationen, Informationen zu neuen Technologien und/oder Erfahrungsberichte sowie kurze Code-Beispiele sucht, der sollte sich auf .NET Heute umsehen. Zusätzlich finden sich eine Menge Foren, wie die MSDN Foren.

Zu guter Letzt finden sich viele Personen aus der Community, die doch meistens ein offenes Ohr für den .NET Nachwuchs besitzen. Wer ein wenig guten Willen zeigt, wird sicher nicht abgewiesen.

2.1.2.Wo beginne ich als Anfänger?

Diese Frage wird immer wieder gestellt. Womit soll begonnen werden, wie beginnt man ein Projekt, was ist zu beachten und viele weitere Fragen warten darauf, beantwortet zu werden.

Nun, im Grunde ist es nicht ganz so einfach und Anfänger haben oft das Gefühl, von der Informationsflut quasi erschlagen bzw. ertränkt zu werden. Doch es ist nicht ganz so kompliziert.

Zu unterscheiden gilt:

- Kann ich alles, um ein bestimmtes Projekt erfolgreich abzuschließen.
- Welches Wissen fehlt mir?
- Wie plane ich ein Projekt von A bis Z?

Die erste Frage muss jeder für sich selbst beantworten. Wenn man etwas nicht kann, muss einfach die notwendige Zeit aufgewandt werden. Komme es wie es wolle. Viele Tests, viele Dokumentationen lesen und mit vielen Menschen sprechen. Das quasi offene Geheimrezept.

Es ist zudem sehr wichtig zu wissen, welche Möglichkeiten man nicht hat, wo Wissen zu diversen Themengebieten fehlt und vor allem natürlich: Wie komme ich an dieses Wissen.

Zur dritten Frage gibt es jede Menge Bücher zum Thema Projektmanagement. Hier möchte ich mit meinen Ergüssen keine Verwirrung stiften. Eher besser ein allgemein anerkanntes Buch schnappen und los geht's.

Im Internet lassen sich zahlreiche Ressourcen – vor allem auch für Anfänger – finden. Es macht sich bezahlt, diese zu konsultieren.

Hier ein paar Starthilfen:

- Galileo Computing - <openbooks>
- The CodeProject
- Microsoft CodePlex
- NET Heute

2.1.3.Serialisierung und Versionierung

Bei Verwendung der .NET Serialisierung stellt sich bei einer Änderung von zu serialisierenden Objekten oft die Frage, wie diese auf der Gegenseite behandelt werden. Um Probleme beim Deserialisieren auf der Gegenseite zu vermeiden, können die zusätzlichen Eigenschaften mit unterschiedlichen Attributen versehen werden, um dieser Falle Herr zu werden.

Hier die einzelnen Attribute und welche Bedeutung ihnen zukommt:

[NonSerialized()]
Dieses Attribut gibt an, dass die entsprechende Eigenschaft nicht serialisiert wird.

[OptionalField]
Damit wird die entsprechende Eigenschaft als optional gekennzeichnet. Sendet nun beispielsweise eine ältere Anwendungsversion das Objekt serialisiert zu einer neueren Version, wird das Fehlen der Eigenschaft einfach ignoriert.

[OnDeserializing]
Dieses Attribut wird nicht bei Eigenschaften gesetzt, sondern nur bei einer Methode - und zwar bei ausschließlich einer Methode pro Klasse. Dies wird hauptsächlich für Versionierungszwecken getan. Die Damit gekennzeichnete Methode hat nun die Möglichkeit in den Serialisierungsprozess einzugreifen und die fehlenden Werte zu setzen.

Weitere Informationen und auch Beispiele zu diesen Attributen können durch den MSDN Artikel Version Tolerant Serialization bezogen werden.

2.2.Basis Framework

2.2.1.ICloneable: Etwas unsauber, nicht?

Das Interface `ICloneable` wird für gewöhnlich implementiert, wenn das entsprechende Objekt geklont werden soll. Nun ergibt sich aus meiner Sicht hier ein kleines Problem:

`ICloneable` stellt eine Methode `Clone` zur Verfügung. Diese wird für die Klon-Implementierung verwendet. Nun gibt es aber zwei Möglichkeiten zu klonen:

Deep Copy: Alle Objekte werden dupliziert.

Shallow Copy: Nur Objekte oberster Ebene werden vervielfältigt. Alle anderen Objekte stellen Verweise auf die tatsächlichen Objekte dar.

Das eigentliche Problem dieser Schnittstelle besteht nun in folgender MSDN-Aussage:

Clone can be implemented either as a deep copy or a shallow copy.

Dadurch ist nicht klar definiert, wie die `Clone`-Methode nun zu implementieren ist. Bei der Entwicklung eines Frameworks wirft dies das Problem auf, dass der Verwender dieses Frameworks (abgesehen von der möglicherweise recht dürftigen Dokumentation) nicht weiß, ob sein dupliziertes Objekt nun eine vollständige Kopie darstellt, oder eben nicht.

Aus meiner Sicht empfehle ich daher, `ICloneable` nicht zu verwenden und stattdessen eine eigene Implementierung vorzunehmen, die hier eindeutiger ist. Anbieten würde sich die Erstellung zweier Interfaces nach folgendem Muster:

```
public interface IShallowCloneable<T>
```

```
{
    T ShallowClone();
}
```

Durch die Implementierung des Interfaces `IShallowCloneable` geht nun eindeutig hervor, dass dabei ein Shallow Copy durchzuführen ist.

```
public interface IDeepCloneable<T>
{
    T DeepClone();
}
```

Ebenso verhält es sich beim Interface `IDeepCloneable`.

Zu guter Letzt noch ein Hinweis auf Object.MemberwiseClone(): MemberwiseClone erstellt eine flache Kopie (Shallow Copy) des aktuellen Objektes und könnte für diesen Zweck benutzt werden. Auch hier ein kurzer Auszug aus dem MSDN:

`MemberwiseClone` kann nicht überschrieben und nur über diese oder eine abgeleite Klasse aufgerufen werden. Verwenden Sie eine Klasse, die die `ICloneable`-Schnittstelle implementiert, wenn eine **tiefe** oder **flache** Kopie eines Objekts öffentlich für die Benutzer bereitgestellt werden muss.

Bei diesem Thema scheint man sich also selbst bei Microsoft nicht 100%ig einig zu sein. Daher also mein Rat für eigene Frameworks: Für den Zweck des Kopierens sollten eigene Interfaces bereitgestellt werden. Dadurch können jegliche Zweifel aus dem Weg geräumt werden. In Kombination mit einer eindeutigen Dokumentation sieht sich der Konsument des Frameworks dadurch mit keinerlei Fehl-Information konfrontiert.

2.2.2.Paper zur Common Language Infrastructure (CLI)

Die CLI-Spezifikation gibt es ja schon einige Jahre (klarerweise), doch haben sich die meisten .NET Entwickler die Paper zur CLI gespart. Hier einfach die Links zu den entsprechenden Dokumenten. Ein Durchlesen erhöht auf jeden Fall das Verständnis für .NET und die Arbeitsweise im Hintergrund.

- CLI Partition I - Concepts and Architecture
- CLI Partition II - Metadata Definition and Semantics
- CLI Partition III - Common Intermediate Language (CIL) Instruction Set
- CLI Partition IV - Profiles and Libraries
- CLI Partition V - Annexes

2.2.3.Objekt auf Eigenschaftsänderungen überprüfen

Es kommt dann doch vor, dass es notwendig ist festzustellen, ob sich Eigenschaften eines Objektes verändert haben. Beispielsweise um beim User nachzufragen, ob er die Änderungen auch tatsächlich speichern möchte.

Nun ist es ziemlich öd, sich alle alten Werte zu merken und mit denen vor der Speicherung zu vergleichen ob sich hier Unterschiede finden.

Stattdessen bietet sich die Implementierung des Interfaces `INotifyPropertyChanged` an. Dadurch erhält das eigene Objekt ein Event `PropertyChanged` welches ausgelöst wird, wenn eine der Eigenschaften verändert wurde.

Ein Beispiel aus der Praxis kann dann so aussehen:

```
public class ProgramProperties : BasePropertyClass, INotifyPropertyChanged
{

    #region Members

    private bool _checkUpdateStartup = false;
    private bool _receiveConfigurationsFromServer = false;
    private string _serverUri = null;

    #endregion

    #region Properties

    public bool CheckUpdateStartup
    {
        get { return this._checkUpdateStartup; }
        set
        {
            if (this._checkUpdateStartup != value)
                NotifyPropertyChanged("CheckUpdateStartup");

            this._checkUpdateStartup = value;
        }
    }

    public bool ReceiveConfigurationsFromServer
    {
        get { return this._receiveConfigurationsFromServer; }
        set
        {
            if (this._receiveConfigurationsFromServer != value)
                NotifyPropertyChanged("ReceiveConfigurationsFromServer");

            this._receiveConfigurationsFromServer = value;
        }
    }

    public string ServerUri
    {
        get { return this._serverUri; }
        set
        {
            if (this._serverUri != value)
                NotifyPropertyChanged("ServerUri");

            this._serverUri = value;
        }
    }

    #endregion


    #region INotifyPropertyChanged Members

    public event PropertyChangedEventHandler PropertyChanged;
```

```
    #endregion

    #region Private Members

    private void NotifyPropertyChanged(String info)
    {
        if (PropertyChanged != null)
        {
            PropertyChanged(this, new PropertyChangedEventArgs(info));
        }
    }

    #endregion
}
```

2.2.4.Isolierter Speicher für isolierte Daten!

Durch den Namespace `System.IO.IsolatedStorage` ist es möglich, Daten in einem isolierten Speicher abzulegen. Der Vorteil liegt hier klar auf der Hand: Auf diese Art und Weise können Daten gelesen und geschrieben werden, auf die wenig vertrauenswürdiger Code keinen Zugriff erhalten soll. Somit werden vertrauliche Daten nicht zugänglich gemacht und werden daher zusätzlich abgesichert.

Ein weiterer Vorteil: Diese Variante kann zusammen mit servergestützten Benutzerprofilen verwendet werden. So ist der isolierte Speicher des Benutzers dort verfügbar, wo er sich an der Domäne anmeldet.

Wie kann nun in einen isolierten Speicher geschrieben werden?

```
IsolatedStorageFile isolatedFile =
  IsolatedStorageFile.GetStore
     (IsolatedStorageScope.User | IsolatedStorageScope.Assembly, null, null);

if (isolatedFile != null)
{
    StreamWriter newFile =
    new StreamWriter(
    new IsolatedStorageFileStream
        ("IsolatedStorageTest.txt",FileMode.OpenOrCreate));

    if (newFile != null)
    {
        newFile.WriteLine("This is an IsolatedStorage Test");
        newFile.Close();
    }
}
```

Vorerst wird der isolierte Speicher basierend auf eine Scope-Angabe geladen. Der Scope beschreibt hier die gültigen Ebenen, die auf die Informationen zugreifen dürfen. In der weiteren Folge wird dem `StreamWriter` ein `IsolatedStorageFileStream` übergeben, der die Daten in diesen isolierten Speicher schreibt.

Gelesen werden können die Daten folgendermaßen:

```
StreamReader existingFile =
new StreamReader(
new IsolatedStorageFileStream("IsolatedStorageTest.txt", FileMode.OpenOrCreate));
```

```
if (existingFile != null)
{
    Console.WriteLine(existingFile.ReadToEnd());
    existingFile.Close();
}
```

Die Verwendung von isoliertem Speicher ist - wie oben zu sehen - also doch sehr einfach gehalten. Dementsprechend empfiehlt es sich auch, diesen tatsächlich zu nutzen, wenn sensible Daten im Spiel sind.

Weitere Informationen zu diesem Thema finden sich natürlich im MSDN: Verwenden der isolierten Speicherung.

2.2.5.Design und Implementierung der .NET Generics

Wer sich ausführlicher mit der Implementierung der Generics (Parametric Polymorphism) in der CLR auseinander setzen möchte, dem sei nachfolgender Artikel für den Start (quasi als Entrypoint) ans Herz gelegt:

Design and Implementation of Generics for the .NET Common Language Runtime (PDF, 134 KB)

Weiters empfiehlt sich der MSDN Artikel Introducing Generics in the CLR.

Und zum Schluss noch ein Link, der in die Generics-Programmierung noch ein wenig Licht bringt: Generics (C# Programming Guide).

Wer jetzt noch immer nicht genug von Generics hat, dem sei noch das Projekt Generics.NET auf CodePlex empfohlen.

2.2.6.Funktionsangebot der System.Environment Klasse

Die `System.Environment`-Klasse bietet einige nette Hilfsmittel, die durchaus nützlich sind. Dabei handelt es sich quasi um eine Allerlei-Sammlung. Die angehängte Beispielanwendung (.NET 2.0) gibt alle verfügbaren Daten in einer `ListView` aus. Dabei handelt es sich um folgende Daten:

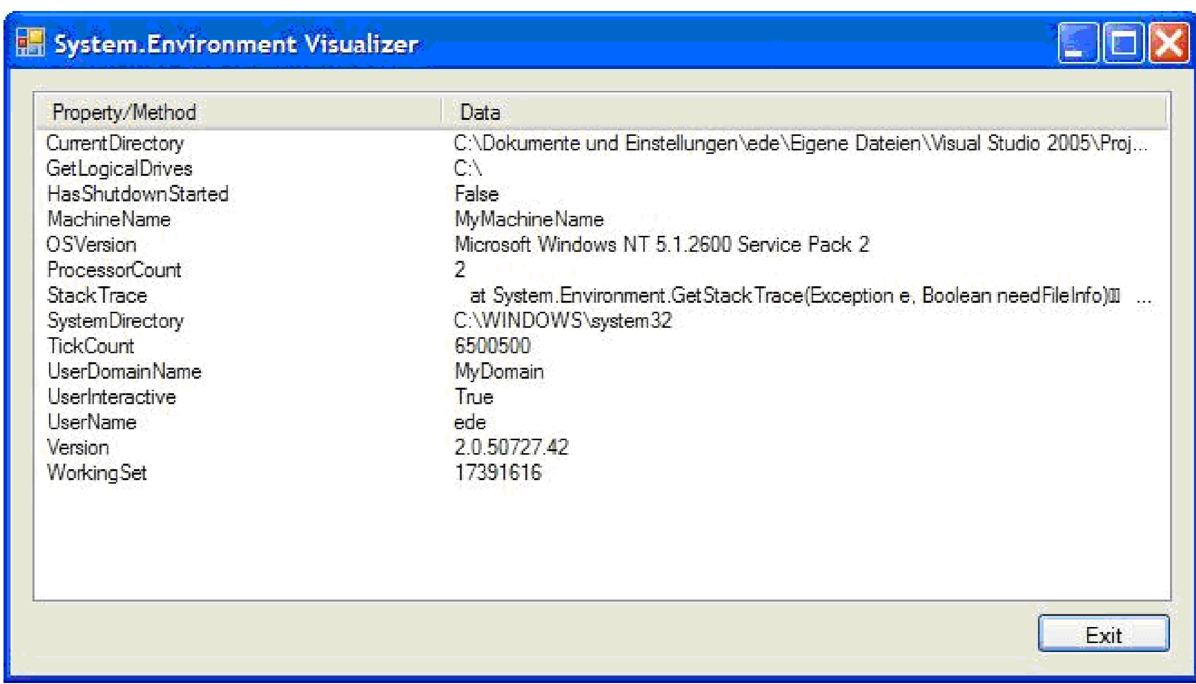


Abbildung 1: System.Environment Visualizer

Zu beachten ist, dass die Klasse auch einige Methoden besitzt, die interessant sind:

- Exit
- ExpandEnvironmentVariables
- FailFast
- GetCommandLineArgs
- GetEnvironmentVariable
- GetEnvironmentVariables
- GetFolderPath
- GetLogicalDrives
- SetEnvironmentVariable

Zusammen also eine nette und vor allem hilfreiche Liste. Eine genaue Beschreibung der genannten Methoden findet sich im MSDN unter System.Environment - Methods.

Hier der Download: System Environment Visualizer

2.2.7.Der Unterschied zwischen const und static readonly!

const
Mit `const` definierte Werte werden mit der Kompilierung als Konstanten ausgewiesen und sind danach nicht mehr veränderbar.

static readonly

Diese "Markierung" erfolgt zur Laufzeit. Damit gekennzeichnete Werte können zur Laufzeit innerhalb der eigenen Klasse modifiziert werden.

`static readonly` wird hauptsächlich verwendet, wenn der gewünschte Typ nicht als `const` deklariert werden darf, oder wenn der Wert erst zur Laufzeit bekannt ist.

2.2.8.Standard-Windows-Sounds abspielen

Seit .NET 2.0 gibt es den Namespace `System.Media` und die Klassen `SystemSound` sowie `SystemSounds`. Damit ist es möglich, Standard-Windows-Sounds abzuspielen um den User auf bestimmte Ereignisse, Fehleingaben etc. hinzuweisen.

Der klassische Beep kann mit nachfolgendem Code abgespielt werden.

```
System.Media.SystemSounds.Beep.Play();
```

Natürlich stehen noch weitere Möglichkeiten zur Verfügung. Einfach mal die entsprechenden Klassen näher betrachten.

Zusätzliche Assemblies müssen hierzu nicht eingebunden werden.

2.2.9.Die wohl häufigste Ausnahme: NullReferenceException

Wer kennt sie nicht, die nachfolgenden Meldungen:

System.NullReferenceException: Object reference not set to an instance of an object
System.NullReferenceException: Der Objektverweis wurde nicht auf eine Objektinstanz festgelegt

Was ist passiert?
Es wurde versucht auf ein Objekt zuzugreifen, welches `null` ist. Im Gegensatz zu Werttypen müssen Referenztypen instantiiert werden. Erst dann kann ein Zugriff darauf stattfinden.

Beispiel

```
MyObject o = null;
o.CallMethod(); // NullReferenceException
```

Hier wurde das Objekt nicht instantiiert und führt daher zu einer `NullReferenceException`.

```
MyObject o = new MyObject();
o.CallMethod(); // keine Exception
```

Da in diesem Beispiel das Objekt instantiiert wurde, wird der Methoden-Aufruf ohne Ausnahme ausgeführt.

Um eine NullReferenceException zu vermeiden ist es sinnvoll, vor dem Zugriff auf ein Objekt zu überprüfen ob es null ist. Ist dem nicht so, kann "weitergearbeitet" werden.

```
MyObject o = null;
if (o != null)
{
    o.CallMethod();
}
```

Damit wird sichergestellt, dass der Code innerhalb der Bedingung nur ausgeführt wird, wenn das Objekt erfolgreich instantiiert wurde. Und so ganz nebenbei fliegen dem Benutzer weit weniger Exceptions um die Ohren.

Womit wir schon beim nächsten Thema wären (Danke Frank für den Hinweis): Exception-Handling. Natürlich muss nicht nur abgefragt werden, ob das Objekt null ist, sondern diese Tatsache ist auch entsprechend zu behandeln. So kann beispielsweise die restliche Aufgabe ohne dieses Objekt nicht korrekt ausgeführt werden. Im Falle eine if-Bedingung ist hier ein entsprechender else-Zweig notwendig. Besser eignet sich ein entsprechender try-catch-Block um diese Ausnahme zu behandeln. So ein Block könnte so aussehen:

```
MyObject o = null;
try
{
    o.CallMethod();
}
catch (NullReferenceException nullException)
{
    // Fehlerausgabe
    // Sonstige Behandlungen
    // Zurücksetzen des Workflows
    // etc.
}
catch (Exception ex)
{
    // eine andere Exception ist aufgetreten
    // diese muss entsprechend behandlet werden
}
finally
{
    // hier Code für Aufräumarbeiten
}
```

2.2.10. Vermeide: if (myBoolean == true)

Oft sieht man Code á la

```
if (myBoolean == true)
```

oder

```
if (myBoolean != true)
```

und jeder hat es zeitweise selbst irgendwo so verwendet.

```
if (myBoolean)
```

oder

```
if (!myBoolean)
```

ist der saubere Weg dies zu tun - und zeugt auch vom Verständnis von booleschen Variablen.

2.2.11.Begriffserklärung: Boxing und Unboxing

Für die einen ein alter Hut, für andere ein Grund um auf die Suche nach Informationen und Erklärungen zu gehen. Nun, ein wenig Licht ins Dunkel kann ich hiermit bringen. Es erfolgt nicht nur eine Erklärung der beiden Begriffe, sondern auch Hinweise wann diese Techniken eingesetzt werden sollen/können und wann dies zu vermeiden sind.

Boxing ist die Konvertierung eines Werttypen in einen Verweistyp. Beispiel:

```
int i = 999; // Werttyp
object o = (object)i; // Konvertierung in einen Verweistyp
```

Unboxing ist die Konvertierung eines Verweistypen in einen Werttyp. Beispiel:

```
object o = 999; // Wert in einem Verweistyp
int i = (int)o; // Konvertierung in einen Wertetype
```

Boxing und Unboxing verursachen zusätzlichen Aufwand, daher sollten diese Operationen vermieden werden. Je häufiger der entsprechende Code-Bereich ausgeführt wird, desto weniger empfiehlt sich die Verwendung.

Es sei an dieser Stelle bemerkt, dass auch der Aufruf von virtuellen Methoden, die Strukturen `von System.Object` erben, unter den Begriff Boxing fallen. (Beispiel: `ToString`).

Empfohlene Vorgehensweisen
Um Boxing und Unboxing zu Vermeiden (und somit auch etwaige Performanceprobleme), empfiehlt es sich, einige Punkte einzuhalten. Hier eine (unvollständige) Liste:

- Werden Strukturen definiert, sollten die Methoden `GetHashCode`, `Equals` und `ToString` überschrieben werden
- Bestehen Methoden die Parameter vom Typ `object` besitzen, die jedoch zur Übergabe von Werttypen verwenden werden, empfiehlt es sich, Überladungen zu definieren, die auf den jeweiligen Werttyp abgestimmt sind.
- Anstatt `object`-Parameter zu verwenden, empfiehlt sich die Verwendung von Generika.

Wann soll Boxing/Unboxing eingesetzt werden
Manche mögen mir jetzt vielleicht widersprechen, aber ich persönlich empfehle Boxing und Unboxing nicht einzusetzen. Selbst bei Code-Teilen die nur selten ausgeführt werden empfiehlt es sich, beispielsweise auf Generika zu setzen oder

Überladungen für die entsprechenden Werttypen zu schaffen. Dadurch erhöhen sich zum einen die Übersichtlichkeit, die Verständlichkeit und vor allem auch die Performance.

2.2.12.C#: Welche Datei repräsentiert einen Prozess

Da manche Prozesse mehrfach ausgeführt werden bzw. werden müssen, ist es manchmal gut zu wissen, welche DLL oder Anwendung der Ursprung eines Prozesses ist. Das nachfolgende Beispiel zeigt, wie eine die Informationen zur gesamten Prozessliste ausgegeben werden. **Achtung**: Die Prozesse `idle` und `System` besitzen kein **main module**.

```
Process[] processes = Process.GetProcesses();
foreach (Process p in processes)
{
    try
    {
        Console.WriteLine(p.ProcessName + " - " + p.MainModule.FileName);
    }
    catch (Exception ex)
    {
        Console.WriteLine(p.ProcessName);
        //Console.WriteLine(ex.Message);
    }
}
```

2.2.13.Zeitmessung einfach gemacht

Soll ein Vorgang performant sein, empfiehlt es sich, unterschiedliche Ansätze zu testen. Diese müssen dann natürlich nicht nur auf Ressourcen-Verbrauch, sondern auch in zeitlicher Hinsicht getestet werden. Dies kann natürlich mit Hilfe von `DateTime` und `TimeSpan` erledigt werden. Das .NET Framework 2.0 enthält hier jedoch auch noch andere Mittel: `Stopwatch` aus dem `System.Diagnostics` Namespace.

`Stopwatch` kann dazu verwendet werden, verbrauchte Zeiten zu messen. `Start` und `Stop` sind die entsprechenden Methoden.

Die Besonderheiten

1. Bei einem `Stop` wird die bereits verbrauchte Zeit nicht zurückgesetzt. D. h. bei einem neuerlichen Start beginnt der Zähler nicht bei Null, sondern inkludiert die bereits verbrauchte Zeit. Mit `Reset` kann der Wert auf Null zurückgesetzt werden.
2. Wem die Präzision und die Auflösung der `Stopwatch`-Implementierung nicht genügt, dem seien die Eigenschaften `Frequency` und `IsHighResolution` ans Herz gelegt.

Weitere Besonderheiten und generelle Informationen finden sich unter [1].

Hier noch ein kleines (sinnloses) Beispiel:

```
private void PerformanceTest()
{
    System.Diagnostics.Stopwatch stopWatch = new System.Diagnostics.Stopwatch();
    stopWatch.Start();
    StringBuilder sb = new StringBuilder();
    for (int i = 0; i < 1000000; i++)
    {
        sb.Append(" ");
    }
    stopWatch.Stop();
    Console.WriteLine("Milliseconds used: " + stopWatch.ElapsedMilliseconds);
}
```

[1] MSDN: Stopwatch-Klasse (System.Diagnostics)

2.2.14.Klassen als obsolet (veraltet) markieren

Vor allem bei der Entwicklung von Frameworks kommt es immer wieder vor, dass bestimmte Klassen durch andere ersetzt werden, da ein anderes Pattern eingezogen wurde, oder einfach nur eine bessere Implementierung gefunden wurde. Nun sollten diese Klassen nicht sofort aus dem Framework entfernt werden - aus Gründen der Kompatibilität zu älteren Versionen.

Hier bietet es sich an, diese Klassen als obsolet zu markieren. Dies kann folgendermaßen erreicht werden:

```
[Obsolete("BeschreibungsText", false)] // Keine Fehlermeldung
public class Test { }

[Obsolete("BeschreibungsText", true)] // Fehlermeldung
public class Test { }
```

Es stellt sich nun lediglich die Frage, wie lange diese Klassen im Framework erhalten bleiben sollten. Ich handhabe dies so, dass obsolete Klassen beim übernächsten Major-Release entfernt werden. Alle Minor-, Build-, oder gar Revision-Versionen werden nicht mit einbezogen. Dies bedeutet nun anhand eines Beispiels:

In der Version 2.0 eines Frameworks wird eine Klasse als obsolete markiert, dann fliegt diese in Version 4.0 tatsächlich hinaus. So bleibt genügend Zeit, auf die neue Klassenstruktur umzustellen.

2.2.15.Verwendung des Namespace Alias Qualifier (::)

Gehen wir von folgendem Code aus:

```
class Program
{
    public class System { }

    const int Console = 1;
    const int number = 2;

    static void Main(string[] args)
    {
        Console.WriteLine(number); // Problem
        System.Console.WriteLine(number); // Problem
```

```
    }
}
```

Hier wird eine Klasse System erstellt, ohne den System-Namespace des .NET Frameworks zu beachten. Zusätzlich gibt es einen privaten Member namens Console. Davon ausgehend können die zwei Aufrufe der Main-Methode nicht mehr funktionieren. Wie dem Abhilfe schaffen?

Dafür gibt es den Namespace Alias Qualifier (::).

```
global::System.Console.WriteLine(number); // Funktioniert
```

Der Sinn? Vor allem bei Frameworks kann es passieren, dass Member eines globalen Namespaces (unabsichtlich) ausgeblendet werden. Durch den Namespace Alias Qualifier können diese jedoch ganz normal verwendet werden.

2.2.16.Der Operator ??

Der ?? Operator ist äußerst hilfreich, wenn Objekte auf null hin überprüft werden soll. Es wird die Linke Seite evaluiert. Ist diese nicht null, wird der linke Wert zurückgegeben, andernfalls der rechte.

Variante mit „?:“:

```
void foo( string v)
{
string r = ( v == null ) ? "null" : v;
}
```

Variante mit dem ??-Operator:

```
void foo( string v)
{
string r = v ?? "null";
}
```

2.2.17.Lese- und Schreibrechte einer Datei überprüfen

Kaum ein Entwickler überprüft bei Dateizugriffe, ob der aktuell angemeldete User auch tatsächlich über die entsprechenden Rechte verfügt. Hintergrund ist wohl, dass die meisten User mit einem Administrator-Account (was zumindest Windows betrifft) angemeldet sind. Hier eine einfache Möglichkeit, die Lese- bzw. Schreibrechte einer Datei abzufragen:

```
public class FileRightsReader
{
    public static bool IsReadable(string filename)
    {
        WindowsIdentity principal = WindowsIdentity.GetCurrent();
        if (File.Exists(filename))
        {
            FileInfo fi = new FileInfo(filename);
            AuthorizationRuleCollection acl =
fi.GetAccessControl().GetAccessRules(true, true, typeof(SecurityIdentifier));
```

```
            for (int i = 0; i < acl.Count; i++)
            {
                System.Security.AccessControl.FileSystemAccessRule rule = 
                (System.Security.AccessControl.FileSystemAccessRule)acl;
                if (principal.User.Equals(rule.IdentityReference))
                {
                    if 
(System.Security.AccessControl.AccessControlType.Deny.Equals
                    (rule.AccessControlType))
                    {
                        if ((((int)FileSystemRights.Read) & 
(int)rule.FileSystemRights) == (int)(FileSystemRights.Read))
                            return false;
                    }
                    else if 
(System.Security.AccessControl.AccessControlType.Allow.Equals
                    (rule.AccessControlType))
                    {
                        if ((((int)FileSystemRights.Read) & 
(int)rule.FileSystemRights) == (int)(FileSystemRights.Read))
                            return true;
                    }
                }
            }
        }
        else
        {
            return false;
        }
        return false;
    }

    public static bool IsWriteable(string filename)
    {
        WindowsIdentity principal = WindowsIdentity.GetCurrent();
        if (File.Exists(filename))
        {
            FileInfo fi = new FileInfo(filename);
            if (fi.IsReadOnly)
                return false;

            AuthorizationRuleCollection acl = 
fi.GetAccessControl().GetAccessRules(true, true, typeof(SecurityIdentifier));
            for (int i = 0; i < acl.Count; i++)
            {
                System.Security.AccessControl.FileSystemAccessRule rule = 
(System.Security.AccessControl.FileSystemAccessRule)acl;
                if (principal.User.Equals(rule.IdentityReference))
                {
                    if 
(System.Security.AccessControl.AccessControlType.Deny.Equals
                    (rule.AccessControlType))
                    {
                        if ((((int)FileSystemRights.Write) & 
(int)rule.FileSystemRights) == (int)(FileSystemRights.Write))
                            return false;
                    }
                    else if 
(System.Security.AccessControl.AccessControlType.Allow.Equals
                    (rule.AccessControlType))
                    {
                        if ((((int)FileSystemRights.Write) & 
(int)rule.FileSystemRights) == (int)(FileSystemRights.Write))
                            return true;
                    }
                }
            }

        }
        else
```

```
        {
            return false;
        }
        return false;
    }

}
```

2.2.18.Mit Dateizugriffs-Rechten arbeiten

In einigen Fällen muss mit den Dateizugriffs-Rechten gearbeitet werden. So muss die ACL (Access Control List) ausgelesen und angepasst werden. Hier ein Code-Snippet, welches den Umgang mit den entsprechenden Klassen aus dem .NET Framework 2.0 zeigt. Kurze Beschreibungen gibt es als Code-Kommentare.

```
// Creating a testfile
Console.WriteLine("Creating XML File");

XmlDocument doc = new XmlDocument();
XmlNode root = doc.CreateElement("root");
doc.AppendChild(root);
doc.Save(@"C:temptest.xml");

Console.WriteLine("Getting FileSecurity");

// Getting Access Control List (ACL) of the file
FileSecurity fSec = File.GetAccessControl(@"C:temptest.xml");

// Get Access Right Type
Type accessRightType = fSec.AccessRightType;
Console.WriteLine("AccessRightType: " + accessRightType.FullName);

// Get Owner of the file
string fileOwner =
fSec.GetOwner(typeof(System.Security.Principal.NTAccount)).Value;
Console.WriteLine("Owner: " + fileOwner + System.Environment.NewLine);

// Get Access Rules of the file
AuthorizationRuleCollection authRuleColl =
fSec.GetAccessRules(true, true, typeof(System.Security.Principal.NTAccount));

// Iterate through all Access Rules
foreach (FileSystemAccessRule rule in authRuleColl)
{
    Console.WriteLine("Control Type : " + rule.AccessControlType.ToString());
    Console.WriteLine("Identity : " + rule.IdentityReference.Value);
    Console.WriteLine("Inheritance Flags: " + rule.InheritanceFlags.ToString());
    Console.WriteLine("Is Inherited : " + rule.IsInherited.ToString());
    Console.WriteLine("Propagation Flags: " + rule.PropagationFlags.ToString());
    Console.WriteLine("File System Right: " + rule.FileSystemRights.ToString());
    Console.WriteLine(System.Environment.NewLine);
}

Console.WriteLine("Adding new Rule");

// Adding a new rule to the file's Access Control List
fSec.AddAccessRule(new
System.Security.AccessControl.FileSystemAccessRule(
System.Security.Principal.WindowsIdentity.GetCurrent().Name,
System.Security.AccessControl.FileSystemRights.Read &
System.Security.AccessControl.FileSystemRights.Write,
System.Security.AccessControl.AccessControlType.Allow));

Console.WriteLine("Setting nun FileSecurity");
```

```
// Save Access Control List
File.SetAccessControl(@"C:temptest.xml", fSec);

Console.ReadKey(false);
```

2.2.19.Assemblies nur zu Reflection-Zwecken laden

Die Assembly-Klasse bietet die Methoden `ReflectionOnlyLoad` und `ReflectionOnlyLoadFrom` um Assemblies in einen eigenen Reflection-Context zu laden.

Damit können Assemblies einfach geladen werden um per Reflection Informationen aus diesen zu beziehen.

Nicht möglich ist damit das Instanzieren von Objekten, ebenso wenig werden Abhängigkeiten automatisch mit geladen. Diese müssen separat nachgeladen werden.

Hinweis: Nur .NET >= 2.0

2.2.20.List<Person> oder doch lieber eine PersonCollection?

In der letzten Zeit ergeben sich immer wieder des Abends Kurzdiskussionen zu bestimmten Themen. Hier nun die Frage bezüglich typisierter Collections (strongly-typed collections) und wie diese in der Praxis anzuwenden sind.

Anstatt Generics via `List<Person>` zu verwenden, würde sich anbieten, eine `PersonCollection` mittels

```
public class PersonCollection : List<Person> { }
```

zu erstellen und diese zu nutzen. Ist das sinnvoll?

Persönlich bevorzuge ich die Generics-Variante, da hier auf den ersten Blick ersichtlich ist, was auch tatsächlich gemeint ist (Lesbarkeit und Verständnis des Codes). Die zweite Variante würde ich vorziehen, wenn zusätzliche Funktionen implementiert werden sollen.

2.2.21.Reflection Geschwindigkeitstest

Da ich mir heute wieder mal einen O/R Mapper genauer angesehen habe (Name der Redaktion bekannt), ist mir ein sehr wesentlicher Punkt aufgefallen. Natürlich arbeiten diese Frameworks mit Reflection. Selten wird jedoch tatsächlich auf Geschwindigkeit gesetzt, so eben das Framework, welches ich heute in die Finger bekommen habe. Daraufhin musste ich gleich einen Test machen.

Der Testfall ist ein sehr einfacher, der im Normalfall keine wesentliche Last verursacht. Es wird aus einer Assembly lediglich ein bestimmter Typ aufgrund seines

Namens geladen und alle Eigenschaften ausgegeben. Die ganze Übung wird 200.000 Mal ausgeführt, was in einer mittelgroßen Anwendung nicht sehr viel ist.

Wenn man davon ausgeht, dass die Engpässe in der Reflection bei quasi rekursiven Durchläufen erst so richtig entstehen, dann sollte mein Testfall nicht sehr gravierend ausfallen. Hier jedoch das Testergebnis auf meinem Rechner:

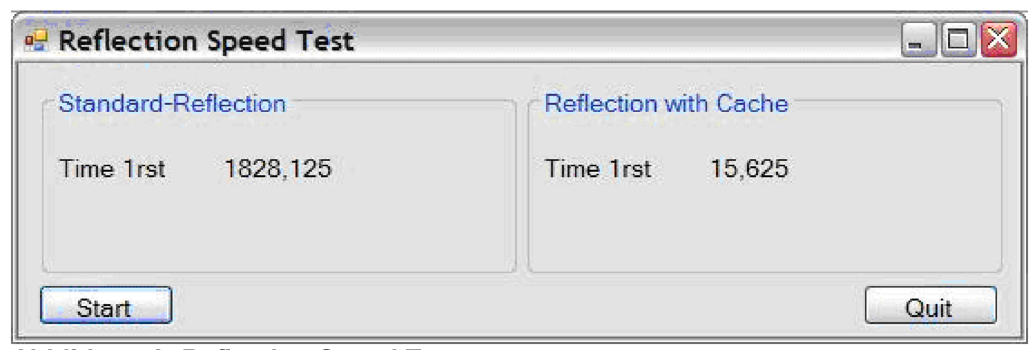


Abbildung 2: Reflection Speed Test

Was genau bedeutet dies? Nun, im ersten Durchlauf wird der nachfolgende Code aufgerufen.

```
for (int i = 0; i < 200000; i++)
{
    Assembly a = Assembly.GetExecutingAssembly();
    if (a != null)
    {
        Type t = a.GetType("ReflectionSpeedTest.Person");
        FieldInfo[] fis = t.GetFields();
        foreach (FieldInfo fi in fis)
        {
            Console.WriteLine(fi.Name);
        }
    }
}
```

Im Vergleich dazu, werden die Feld-Informationen in der zweiten Variante gecached. Das hat zwar den Nachteil, dass der Speicherverbrauch ansteigt, jedoch die Geschwindigkeit um ein Vielfaches erhöht wird - und das bei dieser wirklich sehr einfachen Aufgabe.

```
this.fieldInfos = new Hashtable();

Assembly a = Assembly.GetExecutingAssembly();
if (a != null)
{
    Type t = a.GetType("ReflectionSpeedTest.Person");
    FieldInfo[] fis = t.GetFields();
    foreach (FieldInfo fi in fis)
    {
        this.fieldInfos.Add(fi.Name, fi);
        Console.WriteLine(fi.Name);
    }
}

for (int i = 0; i < 199999; i++)
{
    IDictionaryEnumerator en = this.fieldInfos.GetEnumerator();
    while (en.MoveNext())
    {
        Console.WriteLine(en.Key.ToString());
```



Insgesamt ebenfalls wieder 200.000 Aufrufe. Über 1.800 Millisekunden im Vergleich zu knapp mehr als 15 Millisekunden sprechen hierbei schon Bände. Man bedenke, dass oft aufwändigere Probleme mittels Reflection gelöst werden, daher sollte ein Caching durchaus überlegt werden. An sich sehr einfach, jedoch eine große Wirkung.

2.2.22.MD5-Wert eines Strings

Immer wieder taucht die Frage auf (obwohl an vielen Stellen im Internet auffindbar), wie denn ein String in einen MD5-Hash umgewandelt werden kann. Hier ein kleines Code-Snippet dazu:

```
public static string GetOneWayHash(string val)
{
  byte[] data = System.Text.ASCIIEncoding.ASCII.GetBytes(val);
  MD5 md5 = new MD5CryptoServiceProvider();
  byte[] res = md5.ComputeHash(data);
  return System.Convert.ToBase64String(res, 0, res.Length);
}
```

Ich möchte hierzu jedoch einen sehr wichtigen Punkt loswerden, der oftmals falsch interpretiert wird:

Ein MD5 ist ein **Hashverfahren** und keine Verschlüsselung. Dies bedeutet, dass das Zurückgewinnen des ursprünglichen Wertes im Normalfall nicht möglich ist. Warum nur im Normalfall: Beispielsweise sollten Passwörter nie in ihrem originalen Wert abgespeichert werden. Hierfür bieten sich Hashverfahren ab, die einen Hashvalue errechnen und diesen anstatt des ursprünglichen Passwortes ablegen. Aus dem resultierenden Wert soll es nicht möglich sein, das eigentliche Passwort zu errechnen. Im Gegensatz zur Verschlüsselung. Hier ist es von absoluter Wichtigkeit, die ursprünglichen Daten in derselben Form zurück zu erhalten.

2.2.23.Graphics.MeasureString: Maßeinheit der Rückgabe beeinflussen

Immer wieder taucht die Frage auf, in welcher Einheit die Rückgabe der Methode `MeasureString` (`Graphics`-Object) gehalten ist. Hier die Antwort.

Die Maßeinheit wird über die PageUnit-Eigenschaft des `Graphics`-Objektes beschrieben. Hierzu wird die Enumeration `GraphicsUnit` verwendet. Diese besitzt folgende Werte:

- Display
- Document
- Inch
- Millimeter
- Pixel
- Point

- World

Um also die Breite eines Strings in Pixel zu erfahren, ist vor dem Aufruf von `MeasureString` die Eigenschaft `PageUnit` des `Graphics`-Objektes auf `GraphicsUnit.Pixel` zu stellen.

MSDN: Graphics Class
MSDN: MeasureString Method
MSDN: PageUnit Property

2.2.24.Installierte Grafik-Codecs?

Installierte Grafik-Codecs können mit nachfolgendem Code ermittelt werden:

```
ImageCodecInfo[] encoders = ImageCodecInfo.GetImageEncoders();
foreach (ImageCodecInfo codecInfo in encoders)
     Console.WriteLine(codecInfo.MimeType);
```

2.2.25.JPEG Grafiken verlustlos rotieren

Der nachfolgende Code zeigt, wie JPEG Grafiken (und ebenso auch andere Formate) verlustlos rotiert werden können:

```
Image i = Image.FromFile(this.imageFilename);
ImageCodecInfo usedIC = this.GetEncoderInfo("image/jpeg");

System.Drawing.Imaging.Encoder encoder =
System.Drawing.Imaging.Encoder.Transformation;

EncoderParameters encparams = new EncoderParameters(1);
EncoderParameter encparam =
     new EncoderParameter(encoder, (long)EncoderValue.TransformRotate270);
encparams.Param[0] = encparam;

i.Save("filename.jpg", usedIC, encparams );

i.Dispose();
i = null;
GC.Collect();
```

2.2.26.Die grafische Länge eines Strings mit C# bestimmen

In manchen Fällen (bei der Erstellung von UserControls oder der Verwendung von GDI+) ist es notwendig, die grafische Länge eines Strings zu kennen (also nicht nur die Anzahl der Zeichen). Nachfolgender Code zeigt, wie dies bewerkstelligt werden kann:

```
string test = "This is a test!";

Font font = new Font("Arial", 10.0F);
```

```
Graphics g = this.CreateGraphics();
SizeF sizeInfo = g.MeasureString(test, font);
```

`this` ist in diesem Fall eine Form, kann jedoch genauso gut eine `PictureBox` etc. sein.

2.2.27.Strong named assembly trotz Referenz auf eine ActiveX Komponente

Will man eine Assembly per Strong Name signieren und hält diese Assembly eine Referenz auf eine ActiveX Komponente (beispielsweise AxInterop.SHDocVw), dann kommt es zu folgendem Fehler:

Assembly generation failed — Referenced assembly 'AxInterop.SHDocVw' does not have a strong name

Dieser Fehler ist jedoch recht einfach zu umgehen.

Durchzuführende Schritte

Hierzu benötigen wir den Pfad zur ursprünglichen ActiveX Komponente (hier namentlich als `MyActiveX.dll` geführt). Der Pfad kann mittels der bereits vorhandenen Referenz (Eigenschaften) ermittelt werden. Die ActiveX-Komponente ist nun mittels `tlbimp.exe`.

zu importieren:

```
tlbimp.exe MyActiveX.dll /out:Interop.MyActiveX.dll

/keyfile:MyKeyFile.snk
```

Nach diesem Vorgang ist die vorhandene Referenz auf die Interop-Assembly zu entfernen und die neu erstellte Interop.`MyActiveX.dll` einzubinden. Ein Rebuild sollte nun fehlerlos durchlaufen.

2.2.28.Strong named assembly trotz Referenz auf eine ActiveX Komponente Teil 2

Ebenfalls sehr interessant ist das Tool `AxImp.exe`.

Das **ActiveX Control Importer Tool** konvertiert Typ-Informationen einer COM-Typ-Library in ein Windows Forms Control.

Unter Windows Forms ist es nur möglich Controls zu hosten, die von der Control-Klasse abgeleitet sind. AxImp.exe generiert nun einen Wrapper für ActiveX Komponenten welche unter den Windows Forms verwendet werden können.

Um also ein ActiveX Control hosten zu können, muss ein entsprechendes Wrapper-Control erstellt werden. Dieses muss zudem von AxHost abgeleitet sein. Das Wrapper-Control enthält dabei eine Instanz des zugrunde liegenden ActiveX Controls

und bietet die Möglichkeit darauf zuzugreifen. Alle Methoden, Eigenschaften und Events können genutzt werden.

Um beispielsweise einen entsprechenden Wrapper für das Internet Explorer Control zu erstellen, sind folgende Befehle auszuführen:

```
sn -k shdocvw.snk
AxImp %WINDIR%System32shdocvw.dll /keyfile:shdocvw.snk
```

Die beiden benötigten Dateien `AxSHDocVw.dll` und `SHDocVw.dll` werden daraufhin erstellt.

Für Informationen über `tlbimp.exe` verweise ich auf den ersten Teil [1] dieses Themas.

[1] Informationen über tlbimp.exe

2.2.29. Strong named assembly trotz Referenz auf eine ActiveX Komponente Teil 3

Im zweiten Teil dieser Serie erwähnte ich das Tool `AxImp.exe`. Ich fand jedoch einen wesentlich einfacheren Weg heraus, diese Aufgabe mit Visual Studio 2003 zu erledigen.

Hierzu ist lediglich das Projekt-Eigenschaften-Fenster des aktuellen Projektes zu öffnen. Unter Common Properties/General finden sich die beiden Einträge

- Wrapper Assembly Key File und
- Wrapper Assembly Key Name

Im ersten Feld ist der Pfad zum Keyfile anzugeben, im zweiten ein entsprechender Name. Dadurch werden die Wrapper-Klassen automatisch signiert.

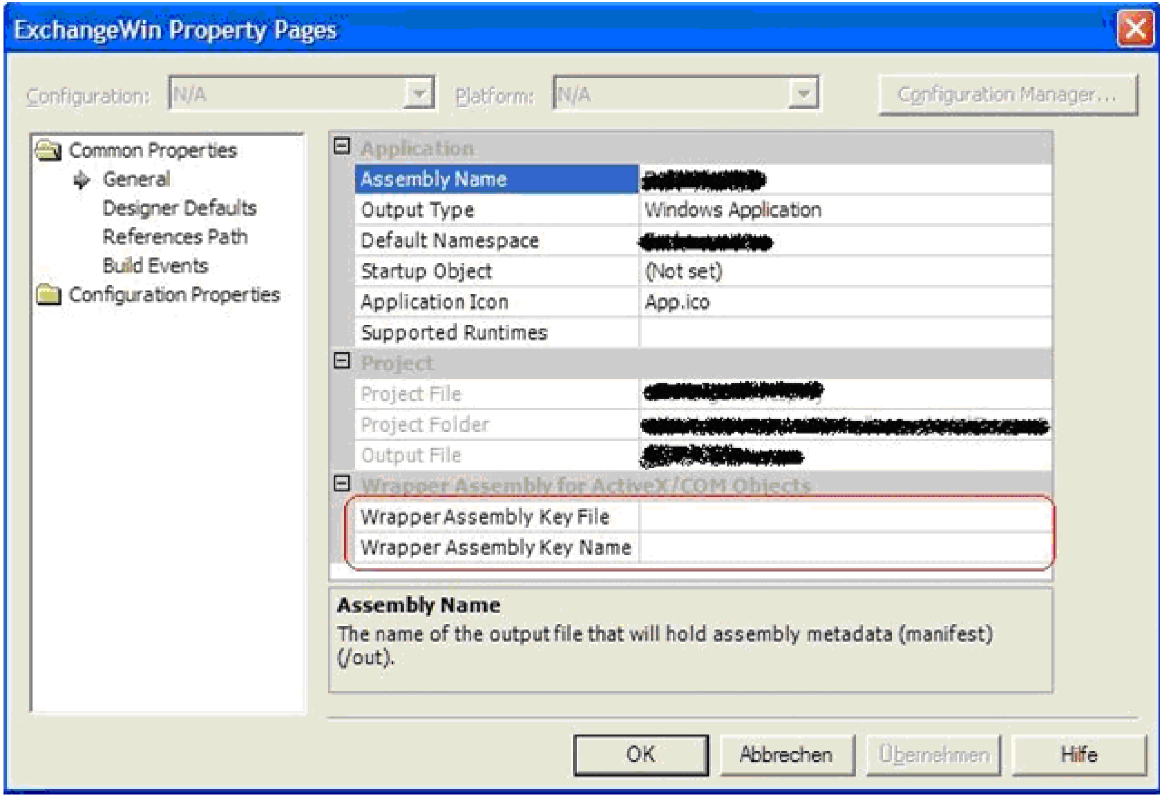


Abbildung 3: Assembly Key File

2.2.30.Object Mapping oder doch lieber DataBinding?

Oft bekommt man in diversen Foren die Frage zu sehen, ob denn nun im eigenen Projekt ein Object Mapping oder doch ein DataBinding verwendet werden soll. Hier ein paar Punkte - aus meiner Sicht - um diese Frage zu beantworten.

Aus meiner Erfahrung sollte Object Mapping dann verwendet werden, wenn untenstehende Fragen mit **Ja** beantwortet werden können.

- Gibt es jede Menge Data-Objects welche auf ebensolche Tabellen gemappt werden sollen und stammen diese auch alle von der gleichen Basis-Klasse ab?
- Könnte der Fall eintreten, dass das zugrunde liegende Datenbank Management System (DBMS) ausgetauscht wird? Sollte das Projekt generell unterschiedliche DBMSs unterstützen?
- Sind für die Entwicklung der Lösung mehr als 15 Manntage notwendig?
- Soll die Lösung von vielen unterschiedlichen Usern eingesetzt werden? (Open Source Projekt, kostenlose Webanwendung)

Können alle Fragen mit einem klaren Ja beantwortet werden, würde ich persönlich zu einem Object Mapping (beispielsweise NHibernate [1]) raten. In anderen Fällen würde ich dann doch eher ein simples DataBinding vorziehen.

Aber Achtung: Immer gründlich die Zukunft im Auge behalten und nicht nur von der

Jetzt-Situation ausgehen. Dinge können sich ändern. Wurde einmal eine Entscheidung getroffen, kann diese meist nur mehr sehr schwer geändert werden.

Bei Anregungen oder einfach Dingen die ich nicht bedacht habe, bitte ich einen Kommentar zu hinterlassen.

[1] NHibernate

2.2.31.WebRequest und SSL Zertifikate

muss man via .NET auf Webseiten zugreifen, die per SSL gesichert sind, kommt es hin und wieder zu Problemen, wenn die Zertifikate nicht verifiziert werden können. Diesem Problem kann man aus dem Weg gehen. Dazu einfach die folgende Klasse implementieren:

```
public class TrustAllCertificatePolicy
: System.Net.ICertificatePolicy
{
  public TrustAllCertificatePolicy()
  {}

  public bool CheckValidationResult(ServicePoint sp,
     X509Certificate cert,WebRequest req, int problem)
  {
    return true;
  }
}
```

Danach ist die Policy nur mehr zu aktivieren. Dies kann folgendermaßen erreicht werden:

```
System.Net.ServicePointManager.CertificatePolicy =
new TrustAllCertificatePolicy();
```

Und schon sollte der Zugriff auf diese Webseiten funktionieren.

2.2.32.C# Beginner: Enumeratoren vs Flags

Enumeratoren sind ja den meisten C# Entwicklern durchaus bekannt. Flags allerdings werden nicht sehr oft eingesetzt. Diese Erfahrung habe ich in diversen Foren gemacht und daher möchte ich dazu ein paar Worte verlieren.

Enumeratoren
Ein Enumerator kann bestimmte vordefinierte Werte enthalten und ist vor allem für Aufzählungen sehr praktisch. Ein Beispiel wäre hierfür der Color-Enumerator. Dieser ermöglicht das einfache Auswählen von Farbwerten. Ein eigenes Beispiel würde wie folgt aussehen:

```
public enum TestEnum
{
  EnumValue0 = 0,
  EnumValue1 = 1
}
```

Für einzelne Items kann hier ein bestimmter Wert definiert werden, muss jedoch nicht. Standardmäßig repräsentiert ein Enumerator einen Int32-Value und beginnt bei 0, außer anders definiert. Eine Zuweisung sieht beispielsweise wie folgt aus:

```
private TestEnum testEnum = TestEnum.EnumValue0;
```

Zu beachten ist, dass testEnum immer nur einen Wert enthalten kann. Dies führt uns nun zur Frage: "Was tun, wenn ich jedoch mehrere Werte speichern möchte?". Ganz einfach:

Flags

Flags können mehrere Werte enthalten. Zuerst jedoch ein Beispiel für die Definition eines Flags:

```
[Flags]
public enum TestFlag
{
  FlagValue1 = 1,
  FlagValue2 = 2,
  FlagValue4 = 4,
  FlagValue8 = 8,
  FlagValue16 = 16
}
```

Im Grunde handelt es sich hierbei um einen Enumerator, dem zusätzlich das Attribut [Flags] verpasst wird. Zusätzlich sind die einzelnen Werte entsprechend des Dualsystems (1, 2, 4, 8, 16, 32, ...) anzugeben, da diese bei Mehrfachauswahl durch ein logisches Oder verknüpft werden.

```
TestFlag testf = TestFlag.FlagValue1;
testf := TestFlag.FlagValue2;
```

In diesem Fall hat testf den Wert 3. Dieser repräsentiert FlagValue1 und FlagValue2. Durch ein

```
testf -= TestFlag.FlagValue1;
```

reduziert sich der Wert auf 2, wodurch auch nur mehr dieses Flag gesetzt ist. Abfragen können nun folgendermaßen gemacht werden:

```
if ( (testf & TestFlag.FlagValue1) > 0) {}
```

Trifft diese Bedingung zu, ist das Flag gesetzt, andernfalls würde das Ergebnis 0 sein und die Bedingung würde folgerichtig nicht zutreffen.

```
if ( (testf & TestFlag.FlagValue1) == 0) {}
```

Auf diese Weise kann festgestellt werden, ob ein bestimmtes Flag nicht gesetzt ist. Trifft die Bedingung zu ist es nicht gesetzt, andernfalls schon.

Fazit

Dies sollte einen kurzen Einblick in die Welt der Enumeratoren und Flags bieten. Flags bieten in vielen Fällen eine einfache Lösung für Mehrfach-Auswahlen und können auch entsprechend in UserControls abgebildet werden, um dem User eine vereinfachte Darstellung zu bieten.

2.2.33. System.IO.Compression - Hilfreich oder doch ein Scherz?

Unter dem .NET Framework 2.0 gibt es ja den *System.IO.Compression*-Namespace. Dieser beinhält Klassen um Dateien zu zippen. Vorgangsweise sieht so aus, dass Daten in einen Stream geschrieben werden (beispielsweise einem GZipStream). Dieser zippt danach die Daten, welche in weiterer Folge in einer Zip-Datei abgelegt werden können.

Dies funktioniert wenn man eine einzelne Datei zippen möchte. Was ist, will ein gesamtes Verzeichnis gepackt werden? In diesem Fall muss der Software-Entwickler seinen eigenen Container entwickeln oder auf vorhandene Bibliotheken von Fremdherstellern zurückgreifen. Bei der Entwicklung eines eigenen Containers kann zwar ein eigenes System entworfen werden, welches auf die Anforderungen zugeschnitten ist, jedoch geht die Kompatibilität zu bestehenden Zip-Anwendungen verloren.

Weiterführende Informationen können unter folgendem Link bezogen werden: .NET System.IO.Compression and zip files

2.2.34. C#-Beginner: Exception-Handling unter C#

Das Thema Exception-Behandlung scheint bei vielen .NET Programmierern noch nicht richtig angekommen zu sein. Immer wieder finden sich in diversen Beispielen und Fragen traurige Konstrukte, die hauptsächlich negative Erscheinungen zu Tage fördern. Also Beispiel sei hier ein neulich gesichteter Code gezeigt werden (nicht kopieren!!!!!):

```
try
{
  StreamReader sr = new StreamReader("path");
  string text = sr.ReadToEnd();
  sr.Close();
}
catch (Exception ex) {}
```

Das Ergebnis? Nun, ist die Datei nicht vorhanden wird eine Exception geworfen und auch abgefangen, aber es passiert damit nichts. Es erfolgt weder eine Meldung an den User, noch ein Log-Eintrag, um etwaige Fehler zu einem späteren Zeitpunkt nachvollziehen zu können. Ein weiterer Effekt ist, dass beim "Testen" durch den Entwickler "alles funktioniert" - was natürlich nicht stimmt.

Nun gut, aber wie soll das Exception Handling dann wirklich umgesetzt werden? Ganz einfach. Der grundlegende try-catch-Block sieht so aus:

```
try {
// Implementierung
} catch (IOException ex) {
// Fehlerbehandlung für IO-Fehler
} catch (Exception ex) {
// Fehlerbehandlung für andere Fehler
} finally {
```

```
// Abschlussarbeiten
}
```

Hier noch eine genaue Beschreibung:

`try`: Im `try`-Teil des gesamten Blockes erfolgt die Implementierung der tatsächlichen Funktion.

`catch`: Hier ist die Fehlerbehandlung zu implementieren. Dies kann realisiert werden, indem die Fehlermeldungen bzw. zusätzliche Einträge in eine Log-Datei geschrieben werden oder eine User-Interaktion verlangt wird. Wie oben gezeigt, können mehrere Exceptions gezielt abgefangen und behandelt werden. Im MSDN finden sich zu allen Methoden auch Angaben darüber, welche Exceptions geworfen werden. Prinzipiell ist mit diesen Exceptions zu arbeiten und nicht direkt mit Exception selbst.

`finally`: Dieser Teil des Blockes wird in jedem Fall ausgeführt, also sowohl nach erfolgreichem Durchlauf des `try`-Teiles, als auch im Falle eines Fehlers. Dadurch bietet es sich an, im `finally`-Block Aufräumarbeiten durchzuführen. Dies kann beispielsweise das Schließen einer Datenbank-Verbindung sein.

Beim `catch`-Block muss nicht zwingend ein Typ angegeben werden. Ist dies nicht der Fall, dann werden sämtliche Exceptions behandelt. Von dieser Schreibweise würde ich jedoch eher abraten:

```
try {

} catch {

}
```

Des Weiteren sollten Exception nicht für die Ablaufsteuerung einer Anwendung verwendet werden. Darunter wird verstanden, dass gezielt auf Exceptions abgefragt wird, um aufgrund des Exception-Typs zu entscheiden, welcher weitere Code anschließend ausgeführt wird. Exceptions sind sehr "teuer". Dies bedeutet, dass dadurch viele Ressourcen verbraucht werden. Daher sind Exceptions auch als solche zu behandeln.

Natürlich besteht auch die Möglichkeit eigene Exceptions zu implementieren. Dies macht vor allem bei der Entwicklung von größeren Frameworks (die auch von anderen Entwicklern benutzt werden) Sinn. Folgendes Beispiel soll eine eigene Exception verdeutlichen:

```
using System;
class MyException : ApplicationException
{
  public MyException(string str)
  {
    Console.WriteLine(str);
  }
}
```

Wie zu erkennen ist, ist von der Basisklasse `ApplicationException` abzuleiten. Der Konstruktor erhält einen Parameter und damit ist die einfachste Variante einer benutzerdefinierten Exception fertig. Weitere Möglichkeiten können aus dem MSDN bezogen werden.

Benutzerdefinierte Ausnahmen können – wie nachfolgend zu sehen – mittels `throw` geworfen werden.

```
throw new MyException("Eine benutzerdefinierte Exception ist aufgetreten");
```

Eine so geworfene Exception muss natürlich auch entsprechend behandelt werden.

Referenzen und weiterführende Artikel:

[1] Exception Management Architecture Guide
[2] Exception Class

2.2.35.C#: Methode mit Parameter via ThreadStart aufrufen

Im heutigen Beitrag zum Thema "C# Beginner" möchte ich ein Beispiel zeigen, wie mittels `ThreadStart` eine Methode inkl. Parameter aufgerufen werden kann.

Dazu ist einfach die entsprechende Methode in eine eigene Klasse mit den notwendigen Eigenschaften auszulagern:

```
private class TestClass {

  private string parameter = null;

  public string Parameter {
    get { return this.parameter; }
    set { this.parameter = value; }
  }

  public void Start() {
    // code goes here
  }

}
```

In der ursprünglichen Klasse wird der Thread wie folgt gestartet:

```
TestClass tc = new TestClass();
tc.Parameter = "test";
Thread t = new Thread(new ThreadStart(tc.Start)).Start();
```

2.2.36.Connectionstrings unter C#, VB.NET

Immer wieder wird nach den richtigen Connectionstrings für die unterschiedlichsten Datenbank-Systeme gefragt. Daher mein Tipp: einfach auf http://www.connectionstrings.com/ nachsehen. Da sollte das meiste zu finden sein.

Und wie wird dieser dann in C# bzw. VB.NET verwendet? Gut, hierfür kann ich ein kleines Beispiel geben (gilt für den Microsoft SQL Server):

C#

```
using System.Data.SqlClient;
...
SqlConnection conn = new SqlConnection();
conn.ConnectionString = "Data Source=(local);" +
"Initial Catalog=MyDatabaseName;" +
"Integrated Security=SSPI";
conn.Open();
```

VB.NET

```
Imports System.Data.SqlClient
...
Dim conn As SqlConnection = New SqlConnection()
conn.ConnectionString = "Data Source=(local);" & _
"Initial Catalog=MyDatabaseName;" & _
"Integrated Security=SSPI"
oSQLConn.Open()
```

Dies gilt nun für den Microsoft SQL Server, der lokal installiert ist. Für einen SQL Server der auf einem anderen Rechner installiert ist, muss lediglich der Connectionstring ausgetauscht und angepasst werden. Die entsprechenden Connectionstrings sind im oben angeführten Link zu finden.

Zusatzinformation: Die Beispiele funktionieren sowohl bei einem SQL Server 2000, als auch beim SQL Server 2005.

2.2.37.Strings unter .NET

Dieser Eintrag soll aufführen, wie Strings unter .NET intern behandelt werden. Strings sind unter .NET immutable. Das bedeutet, dass sie nicht veränderbar sind.

Man nehme das folgende Beispiel:

```
string muh = "muh";
muh += "kuh";
```

Hier sieht das ganze so aus, dass zuerst ein String-Objekt mit dem Value "muh" gebildet wird. Möchten wir an "muh" den String "kuh" anhängen passiert folgendes:

Es wird ein neues String-Objekt am Heap erzeugt. Die Größte (Länge) des String-Objektes beträgt `muh.Length + "kuh".Length`. Nun wird das Ergebnis aus "muh" + "kuh" in das neue String-Objekt kopiert. Das alte String-Objekt „muh“ wird nun für die Garbage Collection freigegeben.

Was sagt uns das jetzt? Es sollte wenn möglich der StringBuilder verwendet werden. Dieser zahlt sich aber erst nach einigen String-Operationen aus, da die Instantiierung des StringBuilders natürlich auch Kosten verursacht. Ich nehme hier als Richtwert 5 bis 7 String-Operationen.

2.2.38.Einführung Garbage Collector

Garbage Collection unter C# sieht im Prinzip folgendermaßen aus:

Der GC wird meist dann angetriggert, wenn der Heap-Speicher aufgebraucht sprich voll ist, manuell gestartet wird, oder wenn ein anderes Programm mehr Speicher benötigt, als aktuell noch frei ist.

Der GC iteriert durch den Heap und markiert zuerst jedes Objekt als Garbage, also setzt es auf "zu verwerfen". Danach wird nochmals jedes Objekt rekursiv überprüft, ob Referenzen darauf zeigen - es also noch benutzt wird. Jedes Mal, wenn ein Objekt vom Garbage Collector besucht wird, wird es als erreichbar markiert. Nach diesem Durchlauf steht nun fest, welche Objekte erreichbar sind und welche nicht. Zyklische Referenzen (also Objekt A referenziert Objekt B, welches Objekt C referenziert, welches wiederum Objekt A referenziert) werden ebenfalls entsprechend behandelt.

Im nächsten Schritt wird der Managed Heap durchlaufen und dieser kompaktiert. Das bedeutet, dass Objekte, die verworfen werden können durch Objekte, die im Heap weiter oben gelagert sind überschrieben werden. Dadurch verkleinert sich der Heap und benötigte Objekte wandern weiter nach unten. Es entstehen also keine "Löcher" Heap (Fragmentierung). Dadurch müssen natürlich auch alle Referenzen vom GC geändert werden. Dies ist natürlich ein aufwändiges Unterfangen und aus diesem Grund sollte der GC manuell nur aufgerufen werden, wenn dies unbedingt notwendig ist.

Wann werden Objekte nicht mehr benötigt? Beispielsweise werden in einer Methode unterschiedliche Objekte instanziert. Wurde diese Methode durchlaufen, werden diese Objekte natürlich nicht mehr benötigt und werden beim nächsten Durchlauf des GC entsprechend behandelt. Sie müssen also nicht explizit auf null gestellt werden. Anders verhält es sich bei Objekten, die "expensive resources" (Dateien, Socket-Verbindungen, Ports, Daten Strukturen etc.) enthalten. Hier bietet .NET die **object finalization** an. Werden also Verbindungen etc. benutzt sollte eine Methode Finalize() vorhanden sein. Diese wird bei der Freigabe des Objektes aufgerufen und somit können verwendete Resourcen auch entsprechend behandelt bzw. tatsächlich freigegeben werden. Hier ist zu beachten, dass die `Finalize()`-Methode der Basisklasse auch aufgerufen wird. Weiters immer einen try-catch-Block darum ziehen.

Eine weitere Möglichkeit besteht durch das `Dispose()`**-Pattern**. Hier ist eine Methode `Dispose()` zu implementieren. Durch den Aufruf von Dispose werden alle teuren Resourcen entsprechend freigegeben - wie auch in der `Finalize()`-Methode muss hier der Code für die Freigabe der Ressourcen manuell eingetragen werden. Die Ressourcen werden durch diesen Aufruf sofort freigegeben - ausser man wartet auf den Garbage Collector.

Die schönere Variante besteht darin, das `IDisposable()`-Interface zu implementieren. Dadurch wird die Methode `Dispose()` vorgeschrieben und muss entsprechend implementiert werden.

Natürlich gäbe es an dieser Stelle noch mehr zu sagen, aber das sollte als kurzer Überblick durchaus reichen.

2.2.39.Wie die benötigten Rechte einer Assembly feststellen?

Mir hat sich die Frage gestellt, wie man möglichst einfach die für die Ausführung einer Assembly notwendigen Rechte herausfinden kann. Unter .NET 2.0 ist dies recht einfach durch das Tool `permcalc` möglich.

Aufruf

```
permcalc -Show norberteder.com.lib.dll
```

Ausgabe

Folgende XML-Datei wird erstellt:

```
<?xml version="1.0"?>
<Assembly>
<Namespace Name="norberteder">
<Namespace Name="com">
<Namespace Name="lib">
<Namespace Name="globalization">
<Type Name="Translator">
<Method Sig="instance string get_DefaultLanguage()" />
<Method Sig="instance void set_DefaultLanguage(string )" />
<Method Sig="instance string get_CurrentLanguage()" />
<Method Sig="instance void set_CurrentLanguage(string )" />
<Method Sig="instance void RegisterGlobalizationFile(string )" />
<Method Sig="instance void Initialize()">
<Demand>
<PermissionSet version="1" class="System.Security.PermissionSet">
<IPermission version="1" class="System.Security.Permissions.FileIOPermission,
mscorlib, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
PathDiscovery="*AllFiles*" />
</PermissionSet>
</Demand>
<Sandbox>
<PermissionSet version="1" class="System.Security.PermissionSet">
<IPermission version="1" class="System.Security.Permissions.FileIOPermission,
mscorlib, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
PathDiscovery="*AllFiles*" />
</PermissionSet>
</Sandbox>
</Method>
<Method Sig="instance string Translate(struct TranslatorSelector , string )" />
<Method Sig="class Translator get_GetTranslator()" />
<Method Sig="void .cctor()" />
</Type>
<Type Name="TranslatorException">
<Method Sig="instance void .ctor(string )" />
<Method Sig="instance void .ctor(string , class Exception )" />
</Type>
</Namespace>
</Namespace>
</Namespace>
</Namespace>
</Assembly>
```

Aus diesen Informationen kann genau ausgelesen bzw. abgefragt werden, welche Berechtigungen mind. gesetzt werden müssen um alle Funktionen nutzen zu können.

2.2.40.Drucken unter C#

Obwohl es ansich relativ viele Ressourcen im Internet zum Thema 'Drucken unter C#' gibt, wird immer wieder die Frage danach aufgeworfen.

Daher an dieser Stelle ein Codebeispiel:

```
PrintDocument printDoc = new PrintDocument();
printDoc.PrintPage += new PrintPageEventHandler(printDoc_PrintPage);
PrintDialog pd = new PrintDialog();
pd.Document = printDoc;
if (pd.ShowDialog() == DialogResult.OK)
{
    printDoc.Print();
}
```

Was genau passiert hier? Zum Ersten wird ein Dokument angelegt. Dieses stellt quasi die Fläche dar, die ausgedruckt werden soll. Zudem wird ein EventHandler auf das Event `PrintPage` gelegt um feststellen zu können, wann genau gedruckt wird.

Durch den `PrintDialog` kann ein Druck-Dialog zur Auswahl des Druckers angezeigt werden.

Durch Aufruf der Methode `Print()` des PrintDocuments wird der Druck gestartet, wodurch das Event `PrintPage` ausgelöst wird. Darin muss nun der zu druckende Inhalt in das erhaltene Graphics-Objekt geschrieben werden:

```
private void printDoc_PrintPage
(object sender, PrintPageEventArgs e)
{
    String textToPrint = "Test-Ausdruck";
    Font printFont = new Font("Arial", 18, FontStyle.Bold);
e.Graphics.DrawString(textToPrint, printFont, Brushes.Black, 10, 25);
}
```

In diesem Fall wird angegebene Text mit in der Schrift Arial, Schriftgröße 18 und fett auf den Koordinaten x = 10 und y = 25 gedruckt.

Es lohnt sich auch, einen Blick auf die `PrintPageEventArgs` zu werfen.

2.2.41.AppDomains und Memory-Überwachung

Folgende Konstellation:

- ein Prozess
- mehrere AppDomains

Dies könnte beispielsweise ein Plugin-System sein, in welchem Plugins zur Laufzeit wieder entladen werden können.

Nun macht es doch in manchen Fällen Sinn, eine Überwachung des Speichers einzuführen. Schließlich muss man in bestimmten Umgebungen wissen, wieviel Speicher durch welches Modul verbraucht wird. Und zwar im Echtbetrieb und nicht in einer Testumgebung.

Durch AppDomains können unterschiedliche Module/Anwendungen/etc. wunderbar

innerhalb eines Prozesses getrennt werden. Eine Kommunikation durch .NET Remoting ist möglich, jedoch ist jede `AppDomain` für sich abgeschottet. Das macht auch durchaus so Sinn. Ein Punkt jedoch wurde anscheinend von Microsoft nicht bedacht:

Den Speicherverbrauch kann ich mit .NET Boardmitteln nur für einen Prozess bestimmen - was aber nicht immer ausreicht. Wenn schon mehrere Anwendungs-Domänen in einem Prozess laufen können, dann sollte der Speicherverbrauch auch entsprechend weit herunter gebrochen werden können. Funktioniert aber nicht.

Ausweg? Nun, man sieht sich den Heap-Speicher auf CLR-Ebene an (siehe CLR Profiler). Damit weiß man nun, wie viele Objekte geladen sind, wie viele Referenzen auf ein einzelnes Objekt zeigen und wie viel Speicher sie verbrauchen. Nachteil: Die Daten zu den Objekten liegen allerdings im Stack. Wie an diesen Speicherverbrauch herankommen? Weiß ich noch nicht... muss es aber auch fast eine Lösung geben...

2.2.42.C# - Daten zwischen zwei Formularen austauschen

Immer wieder wird die Frage gestellt, wie denn Daten zwischen zwei Formularen ausgetauscht werden können.

Hierfür habe ich kurz ein kleines Testprogramm geschrieben, welches genau dies zeigen soll.

Im Projekt sind zwei Formulare zu finden und eine Klasse `DataExchange`. Diese wird von beiden Formularen verwendet und enthält die entsprechenden Daten.

Aber seht euch doch einfach das Projekt an. Bei Fragen kann ich immer noch weiterhelfen ;-)

Download Source DataExchange.zip

Das Projekt wurde unter .NET 1.1 erstellt, sollte aber auch unter 2.0 lauffähig sein.

2.2.43.C# und Eigenschaften

Eigenschaften sind öffentliche Eigenschaften von Klassen, die private Member kapseln. Das klingt jetzt kompliziert, ist es aber nicht:

Prinzipiell ist es so, dass einfache Variablen in Klassen immer privat und daher von außerhalb der Klasse nicht erreichbar sein sollten. In machen Fällen muss aber auf Werte der Klasse (wenn instantiiert, dann Objekt) zugegriffen werden. Dies kann auf mehrere Arten passieren:

- Die Variablen werden mittels **public** als öffentlich markiert
- Es werden Eigenschaften eingeführt

Variante 1 sollte nicht angewendet werden, also bleibt im Endeffekt nur Variante zwei. Wie sieht das anhand eines Sourcecode-Beispieles aus?

```
public class Test {

    private string name = null;

    public string Name {
        get { return this.name; }
        set { this.name = value; }
    }

}
```

Wie an diesem Beispiel zu sehen, wird in der Variable `name` ein Wert gespeichert. Zugegriffen kann auf diesen Wert mittels der Eigenschaft `Name` werden. Hierzu ist zu beachten, dass Eigenschaften (unterschiedlich zu Methoden) kein () nach dem Methodennamen enthalten, also auch keine Parameter übergeben bekommen können. Weiters gibt es die Bereiche `get` und `set`. Im get-Bereich wird der in name gespeicherte Wert zurückgegeben. Der `set`-Bereich dient dazu, den übergebenen Wert in die private Variable zu speichern. Das Schlüsselwort `value` beinhaltet hierbei den übergebenen Wert. Dies sieht außerhalb der Klasse so aus:

```
Test myTest = new Test();
myTest.Name = "mein Name";
Console.WriteLine(myTest.Name)

Consolen-Ausgabe:
mein Name
```

Nun gut, aber welchen Vorteil hat das ganze nun? Ganz einfach. Durch eine Property kann beispielsweise eine Überprüfung der Werte durchgeführt werden. Zum Beispiel könnte im set-Bereich die Länge des überprüften Wertes abgefragt werden. Überschreitet dieser eine bestimmte Vorgabe, wird der Wert nicht zugewiesen und der alte Wert bleibt erhalten.

Ein Fehler, der oft gemacht wird, ist folgender:

```
private string name = null;

public string Name {
    get { return this.Name; }
    set { this.Name = value; }
}
```

Was genau passiert hier? Sowohl im `get`-, als auch im `set`-Bereich wird immer wieder dieselbe Eigenschaft aufgerufen, was in weiterer Folge zu einem Stack-Überlauf und daher zu einer `StackOverflowException` führt. Hier ist wirklich darauf zu achten, auch tatsächlich die private Membervariable anzugeben.

2.2.44.AppDomains und ShadowCopies

Wer mit Anwendungs-Domänen zu tun hat, stößt irgendwann auch auf das Thema ShadowCopy. Hierfür sind allerdings die vorhandenen Informationen rar gestreut. Das MSDN gibt nicht viel her (nur Oberflächliches) und auch sonst ist dafür nicht viel zu finden. Vor allem nicht, wenn es Probleme gibt. Daher hier eine kleine Anleitung, was bei diesem Thema alles zu beachten ist.

Zuerst kurz eine Begriffserklärung:

AppDomain
Eine AppDomain ist im Endeffekt nichts anderes, als der Kontext, in dem eine Anwendung ausgeführt wird.

ShadowCopies
Dies wird zum Beispiel von ASP.NET betrieben. Assemblies, die vom IIS geladen werden sollen, werden in ein `ShadowCopy`-Verzeichnis kopiert und von dort geladen. Die ursprünglichen Assembly-Dateien werden dadurch nicht gelockt und können von neuen Versionen überschrieben werden.

Aber wie kann man nun selbst ShadowCopies realisieren?

Unter der Annahme, dass in einer Anwendung verschiedene AppDomains geladen werden, sind drei Dinge notwendig (sämtliche Einstellungen können über die Klasse `AppDomainSetup` gemacht werden):

1. ShadowCopy aktivieren

`AppDomainSetup.ShadowCopyFiles` muss hier auf true gesetzt werden. Zu beachten ist allerdings, dass es sich bei dieser Property um einen String handelt.

2. Wo sollen die ShadowCopies erstellt werden?

Hier gibt es zwei Eigenschaften, die interessant sind. `AppDomainSetup.CachePath` und `AppDomainSetup.ApplicationName`. `CachePath` stellt das Verzeichnis dar, in welches die Assemblies kopiert werden sollen. `ApplicationName` ist der Name der Anwendung. Das ganze sieht nun so aus, dass im `CachePath` ein Unterverzeichnis mit dem unter `ApplicationName` angegebenen String erstellt wird. Darunter befinden sich dann die ShadowCopies. Wird darauf verzichtet, kommt der eigentlich dafür zuständige Pfad zum Handkuss: `%userprofile%\local settings\application data\assembly`. ShadowCopies in diesem Pfad werden automatisch wieder gelöscht. Werden beide Eigenschaften korrekt gesetzt und damit ein eigener `CachePath` verwendet, muss man sich auch selbst um das Löschen der Dateien kümmern.

3. Was soll kopiert werden?

Natürlich muss auch angegeben werden, welche Assemblies kopiert werden sollen. Dies wird in der Property `AppDomainSetup.ShadowCopyDirectories` eingestellt. Mehrere Pfade werden hier durch ein Semikolon (;) getrennt angegeben. Alle darin befindlichen Assemblies werden dann fürs `ShadowCopy` herangezogen.

Zum Schluss gibt es auch noch ein wenig Sourcecode. Hier (zur Veranschaulichung" mit fixen Pfaden:

```
AppDomainSetup setup = new AppDomainSetup();

setup.ApplicationName = "Test";
```

```
setup.ApplicationBase = AppDomain.CurrentDomain.BaseDirectory;
setup.PrivateBinPath = AppDomain.CurrentDomain.BaseDirectory;
setup.CachePath = @"C:\temp\cache";
setup.ShadowCopyFiles = "true";
setup.ShadowCopyDirectories = Path.Combine
   (AppDomain.CurrentDomain.BaseDirectory,
   Path.DirectorySeparatorChar.ToString());
```

Die Kopien würden also im Verzeichnis `C:\temp\cache\Test` erstellt werden.

2.2.45.Connection-Probleme zu SQL Server 2005 Express?

Eventuell die nette Meldung **provider: Named Pipes Provider, error: 40 - Could not open a connection to SQL Server**?

Die Lösung ist einfach: Im `ConnectionString` wurde einfach der Instanzname bei der Data Source vergessen. Beispielsweise `127.0.0.1\Instanzname` angeben und schon funktioniert alles wie gewollt.

2.2.46.Wieso gibt es keine Assembly.Unload() Methode?

Jeder, der mit Reflection, AppDomains und Assemblies arbeitet, stellt sich früher oder später die Frage, warum die Assembly-Klasse keine `Unload()`-Methode bietet.

Eine Antwort findet sich in Jason Zander's WeWeblog. Hier werden die damit zusammenhängenden Probleme gut und ausführlich beschrieben.

2.2.47.Unable to debug: The binding handle is invalid.

Welch nette Fehlermeldung beim Startversuch einer Anwendung unter dem Visual Studio 2005.

Nach einiger Recherche kam ich dann auf des Rätsels Lösung - und ich war richtig verwundert.

Gestern ging ich meine Dienste am Rechner durch und deaktivierte alles, was ich eigentlich nicht brauche. Darunter auch die Terminal Services. Das war übrigens ein Fehler. Denn Visual Studio 2005 benötigt die Terminal Services um Debuggen zu können. Aber bitte, nicht dass jemand auf die Idee kommt, mich zu fragen, warum das denn so ist.

Merke: Programmiert man mit Visual Studio 2005, niemals die Terminal Services

deaktivieren. Einfach auf Manuell lassen und schon freut sich das Visual Studio wieder.

2.2.48.Dateien vergleichen

Möchte man zwei Dateien vergleichen, kann man leider auf keine fertige Methode im Framework zurückgreifen. Im MSDN findet man allerdings eine Anleitung inkl. Quellcode um so eine Methode selbst zu schreiben. Eigentlich muss man den Quellcode einfach nur kopieren und in das Projekt einfügen.

Nachfolgend der Quellcode:

```
// This method accepts two strings the represent two files to
// compare. A return value of 0 indicates that the contents of the files
// are the same. A return value of any other value indicates that the
// files are not the same.
private bool FileCompare(string file1, string file2)
{
    int file1byte;
    int file2byte;
    FileStream fs1;
    FileStream fs2;

    // Determine if the same file was referenced two times.
    if (file1 == file2)
    {
          // Return true to indicate that the files are the same.
          return true;
    }

    // Open the two files.
    fs1 = new FileStream(file1, FileMode.Open);
    fs2 = new FileStream(file2, FileMode.Open);

    // Check the file sizes. If they are not the same, the files
        // are not the same.
    if (fs1.Length != fs2.Length)
    {
          // Close the file
          fs1.Close();
          fs2.Close();

          // Return false to indicate files are different
          return false;
    }

    // Read and compare a byte from each file until either a
    // non-matching set of bytes is found or until the end of
    // file1 is reached.
    do
    {
          // Read one byte from each file.
          file1byte = fs1.ReadByte();
          file2byte = fs2.ReadByte();
    }
    while ((file1byte == file2byte) && (file1byte != -1));

    // Close the files.
    fs1.Close();
    fs2.Close();

    // Return the success of the comparison. "file1byte" is
    // equal to "file2byte" at this point only if the files are
```

```
        // the same.
    return ((file1byte - file2byte) == 0);
}
```

Den Artikel dazu findet man unter http://support.microsoft.com/default.aspx?scid=kb;EN-US;320348

2.2.49.Ressourcen schonen - Datenbanken besser öffnen und schließen

Viele Programme und insbesondere ASP.NET Seiten verwenden Datenbanken, um Daten zu speichern und zu verwalten. Grundsätzlich sollte jeder Entwickler sorgsam mit Verbindungen zu solchen Ressourcen umgehen, d.h. geöffnete Verbindungen müssen immer geschlossen werden sobald diese nicht mehr benötigt werden. Allerdings ist oft genau das Gegenteil der Fall. Dieser Artikel soll kurz aufzeigen, welche Fehler am häufigsten gemacht werden, und wie man mit einem kleinen Trick diese umgehen kann.
Verbindungen zu solchen Ressourcen wie Datenbanken herzustellen benötigt Zeit. Deshalb arbeitet das .NET Framework mit dem sog. `Connection-Pooling`. Der `Connection-Pool` hält eine gewisse Anzahl von möglichen Verbindungen vor und teilt diese den anfragenden Teilen des Programms zund Dabei wird natürlich darauf geachtet, dass Verbindungen nicht mehr geöffnet sind, bevor sie neu zugeteilt werden. Sobald alle Verbindungen aufgebraucht sind wird eine entsprechende Ausnahme ausgegeben.
Unter normalen Umständen reicht die Anzahl der möglichen Verbindungen aus. Sollte es doch mal zu einem Engpass kommen ist oft die Unachtsamkeit des Entwicklers schuld, der eine Verbindung nicht geschlossen hat. Aber selbst wenn dieser alles richtig gemacht hat, können offene Verbindungen zurückbleiben.
Häufig werden Datenbanken nach folgendem Schema geöffnet und geschlossen:

```
string sql = "SELECT * FROM Tabelle1";
OdbcConnection sqlConn = new OdbcConnection();
OdbcCommand sqlCmd = new OdbcCommand(sql, sqlConn);

sqlConn.Open();
OdbcDataReader sqlReader = sqlCmd.ExecuteReader();
while (sqlReader.Read())
{
    Console.WriteLine(sqlReader["ID"]);
}

sqlReader.Close();
sqlConn.Close();
```

Dieser Code erfüllt seinen Zweck: Verbindung zur Datenbank wird geöffnet, Daten werden gelesen, Verbindung zur Datenbank wird geschlossen. Was aber passiert mit der Verbindung, wenn während des Lesens der Daten eine Exception auftritt? Sie bleibt geöffnet, zumindest solange bis der `GarbageCollector` die Verbindung irgendwann schließt. Tritt dieser Fehler nun öfters auf, verringert sich die Anzahl der möglichen Verbindungen im `Connection-Pool`, bis eine Exception ausgegeben wird.

Häufig wird jetzt ein `Try-Catch-Finally` Konstrukt in die Methode eingebaut:

```
string sql = "SELECT * FROM Tabelle1";

OdbcConnection sqlConn = new OdbcConnection();
OdbcCommand sqlCmd = new OdbcCommand(sql, sqlConn);

try
{
    sqlConn.Open();
    OdbcDataReader sqlReader = sqlCmd.ExecuteReader();
    while (sqlReader.Read())
    {
        int id = Convert.ToInt32(sqlReader["Name"]);
    }

    sqlReader.Close();
}
finally
{
    sqlConn.Close();
}
```

Der Code wirkt jedoch unübersichtlich und oft wird im `finally` Abschnitt trotzdem das Schließen der Verbindung vergessen.
Übersichtlicher Code, der trotzdem im Falle eines Fehlers die Verbindung schließt, erhält man mit sog. „*scopes*“. Ein `scope` ist nichts anderes als ein in geschweifte Klammern eingeschlossener Codeabschnitt. Das Objekt, welches die Verbindung zur Datenbank aufbaut, ist also nur innerhalb dieses *scopes* gültig. Im Code würde das folgendermaßen aussehen:

```
using (OdbcConnection sqlConn = new OdbcConnection()
{
    OdbcCommand sqlCmd = new OdbcCommand(sql, sqlConn);

    sqlConn.Open();
    OdbcDataReader sqlReader = sqlCmd.ExecuteReader();
    while (sqlReader.Read())
    {
        Console.WriteLine(sqlReader["ID"]);
    }
    sqlReader.Close();
    sqlConn.Close();
}
```

Diese Vorgehensweise hat mehrere Vorteile. Zum einen wird der Zugriff auf eine bereits geschlossene Verbindung verhindert, da das entsprechende Objekt außerhalb des scopes nicht vorhanden ist. Zum anderen wird beim Auftreten eines Fehlers in jedem Fall die Verbindung zur Datenbank geschlossen, da der Compiler diesen Code intern in einen try-catch-finally Block übersetzt. Somit ist es auch möglich das manuelle Schließen der Datenbank ganz wegzulassen. Der Übersicht halber wird es in diesem Beispiel aber trotzdem verwendet.

Mit Hilfe des Tool `ildasm.exe` kann man dies anhand der vom Compiler erzeugten Metadaten überprüfen:

```
TestWithScope.Program::Main : void(string[])
Find   Find Next
  .try
  {
    IL_0012:  nop
    IL_0013:  ldloc.0
    IL_0014:  ldloc.1
    IL_0015:  newobj     instance void [System.Data]System.Data.Odbc.OdbcCommand::.ctor(string,
                                                                                class [System.Data
    IL_001a:  stloc.2
    IL_001b:  ldloc.1
    IL_001c:  callvirt   instance void [System.Data]System.Data.Common.DbConnection::Open()
    IL_0021:  nop
    IL_0022:  ldloc.2
    IL_0023:  callvirt   instance class [System.Data]System.Data.Odbc.OdbcDataReader [System.Data]System.D
    IL_0028:  stloc.3
    IL_0029:  br.s       IL_003e
    IL_002b:  nop
    IL_002c:  ldloc.3
    IL_002d:  ldstr      "ID"
    IL_0032:  callvirt   instance object [System.Data]System.Data.Common.DbDataReader::get_Item(string)
    IL_0037:  call       void [mscorlib]System.Console::WriteLine(object)
    IL_003c:  nop
    IL_003d:  nop
    IL_003e:  ldloc.3
    IL_003f:  callvirt   instance bool [System.Data]System.Data.Common.DbDataReader::Read()
    IL_0044:  stloc.s    CS$4$0000
    IL_0046:  ldloc.s    CS$4$0000
    IL_0048:  brtrue.s   IL_002b
    IL_004a:  ldloc.3
    IL_004b:  callvirt   instance void [System.Data]System.Data.Common.DbDataReader::Close()
    IL_0050:  nop
    IL_0051:  nop
    IL_0052:  leave.s    IL_0066
  }  // end .try
  finally
  {
    IL_0054:  ldloc.1
    IL_0055:  ldnull
    IL_0056:  ceq
    IL_0058:  stloc.s    CS$4$0000
    IL_005a:  ldloc.s    CS$4$0000
    IL_005c:  brtrue.s   IL_0065
    IL_005e:  ldloc.1
    IL_005f:  callvirt   instance void [mscorlib]System.IDisposable::Dispose()
    IL_0064:  nop
    IL_0065:  endfinally
  }  // end handler
```

Abbildung 4: Ressourcen schonen - IL-Code

Wichtig ist die Zeile:

```
IL_005f:callvirt instance void[mscorlib]System.IDisposable::Dispose()
```

Dieser Befehl gibt die Anweisung, dass alle offenen Ressourcen geschlossen werden sollen.
Mit Hilfe dieser kleinen Änderung am Code kann man nun davon ausgehen, dass die Verbindung zur Datenbank immer geschlossen wird. Außerdem fängt man weitere mögliche Fehler schon beim kompilieren ab.

PDF Version

2.2.50.Download einer Datei via HTTP

Die Klasse WebClient ermöglicht es einen einfachen Download einer Datei via HTTP durchzuführen. Ein Anwendungsfall wäre z.B. der Download einer Bilddatei, die sich auf einem Webserver befindet.

Der Quellcode für den Aufruf beschränkt sich auf einen Zweizeiler, verpackt in einer Methode:

```
private static void DownloadFile(string url)
    { WebClient webClient = new WebClient(); webClient.DownloadFile(url,
    Path.GetFileName(url)); }
```

In diesem Beispiel wird die Datei heruntergeladen und lokal unter demselben Namen gespeichert.

2.2.51.Dateidownload ohne Filesystem

Ab und an möchte der ASP.NET Entwickler dem User eine zuvor generierte Datei zum Download anbieten. Die normale Vorgehensweise wäre nun die Datei zu erstellen, diese im Filesystem abzuspeichern und dann zum Client zu streamen. muss man diese Datei archivieren, ist dieses Vorgehen völlig OK. Anders sieht es aus, wenn die Datei nur temporär für den Download erstellt wird. In diesem Fall gibt es eine wesentlich bessere Lösung: Datei erstellen und zum Client streamen.
Fürs Erste benötigen wir eine Methode, die unseren Dateiinhalt in einem String speichert. In diesem Beispiel ist das ein einfacher Dummytext, der 10.000 Mal an den String angehängt wird.

```
private byte[] GenerateFile()
     {
         StringBuilder sb = new StringBuilder();

         for (int i = 0; i < 10000; i++)
         {
             sb.Append("Test;Test;Test;Test;Test;Test;Test;Test\r\n");
         }

         string fileContent = sb.ToString();

         return ConvertStringToByteArray(fileContent);
     }
```

Dazu braucht man natürlich noch eine Methode, die den Stream zum Client schickt

```
private void DownloadFile(byte[] file, string filename)
     {
         Response.AddHeader("Content-disposition", "attachment; filename=" +
filename);
         Response.AddHeader("Content-Length", file.Length.ToString());
         Response.ContentType = "application/octet-stream";
         Response.BinaryWrite(file);
         Response.End();
     }
```

Interessant ist jetzt natürlich die Methode `ConvertStringToByteArray()`. Diese konvertiert, wie der Name schon sagt, einen String in ein Byte-Array mit Hilfe der Klasse `UnicodeEncoding`.

```
private byte[] ConvertStringToByteArray(string stringToConvert)
     {
         return (new UnicodeEncoding()).GetBytes(stringToConvert);
```

```
    }
```

Mehr muss man im ersten Moment auch nicht machen. Mit diesen paar Zeilen spart man sich nun den Umweg über das Filesystem und schickt den Stream der Datei direkt zum Client. Das Beispielprojekt kann man hier herunterladen.

2.2.52.Fehlender Namespace

Wahrscheinlich kennt die Shortcuts schon jeder, aber ich bin gerade erst durch Zufall drauf gestoßen.
Wenn man Klassen verwendet deren Namespace noch nicht eingebunden ist, springt man entweder mit den Cursor an den Anfang der Datei und fügt diesen per Hand hinzu, oder man verwendet den Shortcut “Shift + Alt + F10”.

```
private void GenerateFile()
{
    StringBuilder
}
                    using System.Text;
protected void      System.Text.StringBuilder     sender, Ev
{
```

Abbildung 5: Fehlender Namespace 1

Sehr nützlich ist auch “Ctrl + Tab”. Dieser zeigt eine Übersicht aller geöffneten Dateien und Tool Fenster.

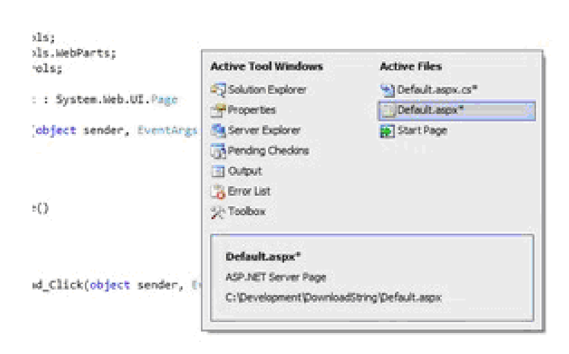


Abbildung 6: Fehlender Namespace 2

2.2.53.Daten aus dem Clipboard in einer Konsolenanwendung verwenden

Möchte man Daten aus dem Clipboard in einer Konsolenanwendung verwenden, muss diese als Single-Threaded laufen. Möglich macht dies das Attribut `STAThread`, welches vor der `Main()`-Methode plaziert werden muss. In Winforms-Anwendungen wird dieses Attribut standardmäßig eingesetzt.

```
using System;
using System.Windows.Forms;

public class ReadFromClipboard
{
    [STAThread]
    public static void Main()
    {
        IDataObject iData = Clipboard.GetDataObject();
        if (iData.GetDataPresent(DataFormats.Text))
        {
            string str = (String)iData.GetData(DataFormats.Text);
        }
    }
}
```

2.2.54.Url per WebRequest auslesen

Immer wieder taucht in div. Foren und Newsgroups die Frage auf wie man eine URL einlesen und das Ergebnis in einem `string` speichern kann. Vor längerer Zeit habe ich mir dafür eine Methode geschrieben, die genau diese Sache erledigt:

```
public static string GetUrlResponse(string url,string username,string password)
{
    string content = null;

    WebRequest webRequest = WebRequest.Create(url);

    if(username == null || password == null)
    {
        NetworkCredential networkCredential = new NetworkCredential(username,
password);
        webRequest.PreAuthenticate = true;
        webRequest.Credentials = networkCredential;
    }

    WebResponse webResponse = webRequest.GetResponse();

    StreamReader sr = new StreamReader(webResponse.GetResponseStream(),
Encoding.ASCII);
    StringBuilder contentBuilder = new StringBuilder();
    while (-1 != sr.Peek())
    {
        contentBuilder.Append(sr.ReadLine());
        contentBuilder.Append("\r\n");
    }
    content = contentBuilder.ToString();

    return content.ToString();
}
```

Der Aufruf sieht wie folgt aus:

```
string responseFromUrl = GetUrlResponse("http://www.google.de", null, null)
```

2.2.55.ApplicationPath in C# Konsolenanwendungen

Ab und zu benötigt man mal den aktuellen Pfad, indem die Konsolenanwendung ausgeführt wird. In Winforms Anwendungen bekommt man dieses über

Application.StarupPath zurück geliefert. In der Konsolenanwendung fehlt diese Klasse allerdings. Damit ich trotzdem den aktuellen Pfad auslesen kann, habe ich folgende Zeilen geschrieben:

```
static string ApplicationPath
{
      get
      {
          return
Path.GetDirectoryName(System.Reflection.Assembly.GetExecutingAssembly().Location);

      }
 }
```

2.2.56.Read From Clipboard

Vielleicht ganz nützlich. Folgende Funktion liest den Inhalt der Zwischenablage aus und gibt ihn als String zurück:

```
private string ReadFromClipboard() {
//Gibt den Inhalt aus dem Clipboard zurück

string strTextFromClipboard = "";
IDataObject iData = Clipboard.GetDataObject();
if(iData.GetDataPresent(DataFormats.UnicodeText)) {
                string strChar = (String)iData.GetData(DataFormats.UnicodeText);
                strTextFromClipboard = strChar;
  }
  return strTextFromClipboard;
}
```

2.2.57.Parsing Dates and Times in .NET

Wie Datums- und Uhrzeitwerte geparst werden können, zeigt der nachfolgende Artikel:
http://www.stevex.org/dottext/articles/916.aspx

2.2.58.String formatting in C#

Ein interessanter Beitrag zum Thema String-Formatierung unter C#:

http://www.stevex.org/dottext/articles/158.aspx

2.2.59.IsolatedStorage (Computerspeicher) verwalten

Wer Konfigurationen oder andere Informationen in den Isolated Storage schreibt, dem wird das Isolated Storage Tool (storeadm.exe) bekannt sein. Wem nicht, der sollte es sich genauer ansehen.

Es handelt sich dabei um eine Konsolen-Anwendung, mit der der Inhalt des

IsolatedStorage angezeigt oder gelöscht werden kann. In manchen Fällen durchaus hilfreich.

Das Tool gibt es - wie auch den isolierten Speicher - seit .NET 2.0 und kann einfach über die Visual Studio 2005 Commandline aufgerufen werden.

Wer Informationen dazu benötigt, der sollte sich folgende Links genauer ansehen:

Einführung in die isolierte Speicherung
Szenarien für die isolierte Speicherung

2.2.60.Prozess-Output via C# anzeigen

Hin und wieder kommt es dann doch einmal vor, dass via .NET andere Prozesse aufgerufen werden und deren Output auf die Standard-Ausgabe eingelesen werden soll. Dazu gibt es mehrere Möglichkeiten.

Die erste Variante liest alle Daten aus der Standard-Ausgabe ein, wenn der Prozess fertig abgelaufen ist:

```
ProcessStartInfo psi = new ProcessStartInfo("MyPathMyApp.exe");
psi.RedirectStandardOutput = true;
psi.UseShellExecute = false;

Process p = new Process();

p.StartInfo = psi;
p.Start();
StreamReader sr = p.StandardOutput;
p.WaitForExit();
string output = sr.ReadToEnd();
sr.Close();
Debug.WriteLine(output);
```

Manchmal dauert der Prozess selbst länger und es ist wichtig, die Daten dann zu bekommen, wenn diese auf der Standard-Ausgabe geschrieben werden. Hierzu kann seit .NET 2.0 mit dem Event `OutputDataReceived` gearbeitet werden. Das sieht so aus:

```
private void StartProcess()
{
    ProcessStartInfo psi = new ProcessStartInfo("MyPathMyApp.exe");
    psi.RedirectStandardOutput = true;
    psi.UseShellExecute = false;
    Process p = new Process();

    p.OutputDataReceived +=
        new DataReceivedEventHandler(p_OutputDataReceived);
    p.StartInfo = psi;
    p.Start();
    p.BeginOutputReadLine();

    p.WaitForExit();
}
```

```
void p_OutputDataReceived(object sender, DataReceivedEventArgs e)
{
    Debug.WriteLine(e.Data);
}
```

In diesen Fällen werden die Informationen immer nur als Debug-Informationen geschrieben. Sie können aber natürlich auch anderweitig weiterverwendet werden.

Hier noch ein kleiner Hinweis: Dadurch wird alles abgefasst, was auf die Standard-Ausgabe geschrieben wird. Liegt also beispielsweise eine Windows-Forms-Anwendung vor, die zwecks Debugging-Informationen (siehe Visual Studio) viele Informationen via `Console.WriteLine` schreibt und das damit verteilt wird, können diese Informationen dadurch ebenfalls ausgelesen werden. Das sollte man sich in manchen Fällen schon ganz gut überlegen (siehe `Debug.WriteLine`).

2.2.61. Einstellungen einfach speichern und laden

Wer beispielsweise viele unterschiedliche Konfigurationen speichern muss (oder auch andere Daten), dem wird es nicht allzu einfach gemacht. Zum einen bietet sich eine Datenbank an (zusammen mit einem O/R Mapper). Wer keine Datenbank verwenden möchte (aus unterschiedlichsten Gründen), der muss sich anders behelfen. Serialisierung ist eine Möglichkeit. Da ich nun genau diesen Fall hatte, schrieb ich mir einen kleinen Manager der diese Aufgabe für mich übernimmt.

Implementiert ist die Klasse als **Singleton** und kann mit allen serialisierbaren Typen umgehen:

```
/// <summary>
/// Read and write Configuration files. Uses the binary formatter.
/// </summary>
public class ConfigurationManager
{
    #region Private Members

    private static object _lockObject = new object();
    private static ConfigurationManager _instance = null;

    private string _configPath = null;

    #endregion Private Members

    #region Properties

    /// <summary>
    /// ConfigurationPath is used to store the configuration
    /// </summary>
    public string ConfigurationPath
    {
        get { return this._configPath; }
        set { this._configPath = value; }
    }

    #endregion Properties

    #region ctor

    /// <summary>
    /// Private Constructor -> Singleton
```

```
/// </summary>
private ConfigurationManager() {}

#endregion ctor

#region Public Methods

/// <summary>
/// Creates a new instance if there wasn't already
/// one created, else the available instance will be
/// returned
/// </summary>
/// <returns>ConfigurationManager instance</returns>
public static ConfigurationManager GetInstance()
{
    lock (_lockObject)
    {
        if (_instance == null)
            _instance = new ConfigurationManager();
        return _instance;
    }
}

/// <summary>
/// Read configuration. The typename is used as the filename
/// </summary>
/// <typeparam name="T">Any serializable type</typeparam>
/// <param name="configuration">Serializable object</param>
/// <returns>Given type T</returns>
public T Read<T>(T configuration)
{
    return Read<T>(configuration.GetType().Name);
}

/// <summary>
/// Read configuration
/// </summary>
/// <typeparam name="T">Any serializable type</typeparam>
/// <param name="configuration">Serializable object</param>
/// <returns>Given type T</returns>
public T Read<T>(string configuration)
{
    if (this._configPath == null
      || !Directory.Exists(this._configPath))
        throw new Exception(
        String.Format(
        "ConfigurationManager: No valid path given: {0}",
        this._configPath ?? "<not set>"));

    string filename =
      Path.Combine(this._configPath, configuration) + ".config";

    if (!File.Exists(filename))
        throw new Exception(
        String.Format(
        "ConfigurationManager: Configuration {0} doesn't exist",
        filename));

    BinaryFormatter bf = new BinaryFormatter();

    FileStream fs = new FileStream(filename, FileMode.Open);

    T tempObject = (T)bf.Deserialize(fs);

    fs.Close();

    return tempObject;
}

/// <summary>
```

```
    /// Write Configuration. The typename is used as the filename
    /// </summary>
    /// <typeparam name="T">Any serializable type</typeparam>
    /// <param name="configuration">Serializable object</param>
    public void Write<T>(T configuration)
    {
        this.Write(configuration, configuration.GetType().Name);
    }

    /// <summary>
    /// Write Configuration.
    /// </summary>
    /// <typeparam name="T">Any serializable type</typeparam>
    /// <param name="configuration">Serializable object</param>
    /// <param name="filename">Filename to be used</param>
    public void Write<T>(T configuration, string filename)
    {
        if (this._configPath == null
          :: !Directory.Exists(this._configPath))
            throw new Exception(
            String.Format(
            "ConfigurationManager: No valid path given: {0}",
            this._configPath ?? "<not set>"));

        string file =
          Path.Combine(this._configPath, filename) + ".config";

        BinaryFormatter bf = new BinaryFormatter();

        FileStream fs =
          new FileStream(file, FileMode.OpenOrCreate);

        bf.Serialize(fs, configuration);

        fs.Close();
    }

    #endregion Public Methods
}
```

Wie kann dieser Manager nun eingesetzt werden? Zuerst muss ein serialisierbares Objekt erstellt werden:

```
[Serializable]
public class Package
{
    public string PackagePath = null;
    public ArrayList DirectoryList = new ArrayList();
    public ArrayList FileTypeList = new ArrayList();
    public ArrayList FilesList = new ArrayList();
}
```

Diese Klasse ist serialisierbar und kann nun mit dem **ConfigurationManager** verwendet werden. (Der Name ConfigurationManager wird von mir für das Speichern von Konfigurationen verwendet. Grundlegend können alle serialisierbaren Typen damit gespeichert bzw. geladen werden.)

Und so geschieht in weiterer Folge der Aufruf:

```
ConfigurationManager cm = ConfigurationManager.GetInstance();
cm.ConfigurationPath = "MyPath";

Package pcp = new Package();
pcp.PackagePath = "Another Path";
cm.Write(pcp, "MyFile.config");
```

```
Package originalPackage = cm.Read("MyFile.config");
```

Anzumerken ist, dass der Pfad des ConfigurationManagers nur einmal gesetzt werden muss, also nicht bei jedem Aufruf (außer die Konfigurationen werden in unterschiedlichen Verzeichnissen abgelegt).

2.2.62.Unbehandelte Ausnahmen (Unhandled Exceptions)

Die Frage nach einer **globalen Behandlung**] von aufgetretenen **Ausnahmen** - und speziell von unbehandelten Ausnahmen) wird oft gestellt. Als mögliche Realisierung wird oft das folgende Beispiel gebracht (Ausgangspunkt ist hier eine Konsolenanwendung):

```
class Program
{
    static void Main(string[] args)
    {
        AppDomain.CurrentDomain.UnhandledException +=
            new UnhandledExceptionEventHandler
            (delegate(object o, UnhandledExceptionEventArgs
            eventArgs)
        {
            Console.WriteLine("Unhandled Exception: " +
                eventArgs.ExceptionObject);
        });
    }
}
```

Verschwiegen (ob bewußt oder aus Nichtwissenheit) wird meist, dass über diesen Weg zwar viele Ausnahmen abgefangen werden können, diese Möglichkeit jedoch nicht als "Exception-Handler" zu verwenden ist.

Warum nicht? Vorerst werden hiermit **nicht behandelnden Exceptions** mitgeteilt. Würde es sich um einen Handler handeln, wären die Exceptions nicht unbehandelt. (Achtung: Rekursions-Alarm!)

Weiters lassen sich einige Exceptions hiermit nicht behandeln, da sie via try-catch nicht abgefangen werden können (seit .NET 2.0: `StackOverflowException`). Zusätzlich ist zu beachten, dass es einen groben Unterschied macht, ob manuell (zwecks Testvorgang) eine StackOverflowException geworfen wird, oder tatsächlich eine echte auftritt. Im ersteren Fall könnte die Exception behandelt werden, im zweiteren Fall wird die Anwendung sofort terminiert.

Zusätzlich sind derartige Verhaltensweise idealerweise in einem Release-Build zu testen, da es Unterschiede zu Debug-Builds gibt und für gewöhnlich Release-Builds ausgeliefert werden.

Das bedeutet: Die einzige sinnvolle Einsatzvariante liegt darin, den aktuellen Anwendungsstatus zu speichern bzw. die nicht behandelnden Exceptions in eine Logdatei aufzuzeichnen. Denn die Anwendung wandert ohnehin über den Jordan (sofern `eventArgs.IsTerminating` wahr ist).

2.2.63.Liste der installierten ADO.NET Provider abrufen

Möchte man überprüfen ob ein bestimmter ADO.NET Provider am System registriert ist, kann dies mit einigen Zeilen Code erledigt werden. Das nachfolgende Beispiel liefert eine Liste aller verfügbaren ADO.NET Provider.

Zu beachten ist an dieser Stelle, dass Provider, die über die `App.Config` eingebunden wurden, hier ebenfalls mit aufgelistet werden.

```
private List GetDbFactoryClasses()
{
    List factClasses = new List();
    DataTable dt = DbProviderFactories.GetFactoryClasses();
    if (dt != null)
    {
        foreach (DataRow dr in dt.Rows)
        {
            factClasses.Add(dr[2].ToString());
        }
    }
    return factClasses;
}
```

Eine Überprüfung ist nicht nur in Problemfällen sinnvoll, sondern sollte auch vor dem Laden eines Providers über `DbProviderFactory` geschehen. Dadurch kann eine Exception vermieden und der Benutzer entsprechend benachrichtigt werden (oder Eintrag in eine Logdatei und ähnliches).

Ein Provider kann folgendermaßen dynamisch geladen werden:

```
// Nur der Vollständigkeit halber
// Wird normalerweise aus einer Konfiguration gelesen

string strProviderInvariantName = "System.Data.SqlClient";
string strConnectionString = "...";


// Laden des Providers
// Erstellen der Datenbank-Verbindung
// Zuweisung des notwendigen ConnectionStrings

DbProviderFactory dbFactory =
    DbProviderFactories.GetFactory(strProviderInvariantName);
DbConnection dbConnection = dbFactory.CreateConnection();
dbConnection.ConnectionString = strConnectionString;
```

Wie bereits in den Code-Kommentaren angemerkt, werden diese Einstellungen normalerweise aus einer Konfiguration geladen oder in einer anderen Form an die Anwendung übergeben. Der Vorteil liegt darin, dass die Anwendung damit mit einem beliebigen ADO.NET Provider ausgeführt werden kann.

2.2.64.String in eine XmlNode konvertieren

Es kommt vor, dass XML-Daten in einem String vorhanden sind und man daraus eine XmlNode erstellen muss. Dies ist ja an sich kein Problem, da hierzu die InnerText-Eigenschaft gesetzt werden kann. Was aber, wenn der String eine Node inklusive einiger ChildNodes enthält? Hier ein einfacher Weg die Konvertierung durchzuführen.

```
XmlDocument doc = new XmlDocument();
doc.Load("categories.xml");
XmlNode xnRoot = doc.SelectSingleNode("/categories");

XmlTextReader xtr =
    new XmlTextReader(new StringReader(category.ToXml()));

XmlNode xnCategory = doc.ReadNode(xtr);
xnRoot.AppendChild(xnCategory);
```

Wie zu erkennen ist, wird dazu der XmlTextReader verwendet, da dadurch einfach Xml-Fragmente eingelesen werden können. Mit Hilfe der Methode ReadNode des XmlDocument-Objektes kann nun ein Node generiert werden, welcher anschließend dem Dokument hinzugefügt wird. Anzumerken ist auch, dass die Methode ToXml des Objektes category ein entsprechendes XML-Fragment aus dem besagten Objekt generiert und als String zurück liefert.

2.2.65. Extern aliases: Namespace aus verschiedenen Assemblies nutzen

Das .NET Framework < 2.0 unterstützte lediglich eine einzelne Namespace-Hierarchie. In diese wurden die Typen der referenzierten Assemblies geladen und konnten anschließend verwendet werden. Dies ist allerdings eher unschön, wenn beispielsweise die gleiche Assembliy, allerdings in unterschiedlichen Versionen, verwendet werden muss oder eine benötigte Assembly zufällig denselben Namespace benutzt wie eine ganz andere. Mit **extern aliases** ist dieses Problem zu lösen.

Gegeben sind zwei Assemblies.

Assembly a1.dll

```
namespace N
{
    public class A {}
    public class B {}
}
```

Assembly a2.dll

```
namespace N
{
    public class B {}
    public class C {}
}
```

Bei nachfolgendem Programm würde es nun zu einem entsprechenden Fehler gelangen:

```
class Test
{
    N.A a; // Ok
    N.B b; // Fehler
    N.C c; // Ok
}
```

Kompilieren wir nun den Code mit folgender Anweisung:

```
csc /r:a1.dll /r:a2.dll test.cs
```

Die Typen aus den Assemblies `a1.dll` und `a2.dll` werden in der **Global Namespace Hierarchy** abgelegt, wodurch ein Fehler entsteht: Der Typ `N.B` existiert in beiden Assemblies. An dieser Stelle kommen die **extern aliases** zum Einsatz:

```
extern alias X;
extern alias Y;

class Test
{
    X::N.A a;
    X::N.B b1;
    Y::N.B b2;
    Y::N.C c;
}
```

Es werden die beiden Aliase `X` und `Y` deklariert, aber die Defintionen der Aliase sind extern abgelegt. Der Kompilierungsaufruf muss entsprechend abgeändert werden und sieht so aus:

```
csc /r:X=a1.dll /r:Y=a2.dll test.cs
```

Unter Verwendung von Visual Studio können die Aliase in der Eigenschaft `Aliases` der jeweiligen Referenz angegeben werden.

2.2.66.Mit IComparable nach mehreren Eigenschaften sortieren

Nehmen wir als Beispiel eine Highscore-Liste. Hierfür müssen zwei Werte sortiert werden:

- der erreichte Score (absteigend)
- der Name des Spielers (aufsteigend)

Mit Hilfe der Schnittstelle `IComparable` kann dies einfach erledigt werden. Hierfür benötigen wir die Datenklasse `Highscore`, welche besagtes Interface implementiert:

```
using System;
```

```
using System.Collections.Generic;
using System.Text;

namespace HighscoreTest
{
    public class Highscore : IComparable
    {
        private string _name = String.Empty;
        private int _score = 0;

        public string Name
        {
            get { return this._name; }
            set { this._name = value; }
        }

        public int Score
        {
            get { return this._score; }
            set { this._score = value; }
        }

        public Highscore(string name, int score)
        {
            this._name = name;
            this._score = score;
        }

        public override string ToString()
        {
            return this._name.PadRight(20, ' ') + this._score;
        }

        #region IComparable Members

        public int CompareTo(Highscore other)
        {
            if (this._score.CompareTo(other._score) == 0)
                return this._name.CompareTo(other._name);
            else
                return this._score.CompareTo(other._score) * (-1);
       }

        #endregion
    }
}
```

Wie in der Methode `CompareTo` zu sehen ist, ist eine signifikante Eigenschaft auszuwählen. Diese ist die Eigenschaft, die vorrangig sortiert werden soll. In unserem Fall der Score. Sind die zu vergleichenden Werte identisch, wird der Name sortiert.

Getestet werden kann dies mit Hilfe der folgenden Zeilen:

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Collections;
using System.Diagnostics;

namespace HighscoreTest
{
    class Program
    {
        static void Main(string[] args)
        {
            List highscoreList = new List();
```

```
            highscoreList.Add(new Highscore("Norbert2", 17));
            highscoreList.Add(new Highscore("Norbert1", 17));
            highscoreList.Add(new Highscore("Norbert3", 3));
            highscoreList.Add(new Highscore("Karl", 3));

            highscoreList.Sort();

            Debug.WriteLine("Highscore");
            Debug.WriteLine("--------------------");
            foreach(Highscore hscore in highscoreList)
                Debug.WriteLine(hscore);

        }
    }
}
```

Wie zu sehen ist, gar nicht so schwierig und keinesfalls aufwändig.

2.2.67.Eigenschaften und Zugriffsmodifizierer

Man nehme folgendes Interface:

Interface

```
public interface IPerson
{
    string Firstname { get; set; }
    string Lastname { get; set; }
    DateTime Birthday { get; }
}
```

Dieses Interface beschreibt, dass die Eigenschaft `Birthday` nur gelesen werden darf, ein Setzen ist nicht möglich. Um dies aber in der Assembly zu erlauben, in der sich auch das Interface befindet kann folgendes definiert werden:

```
public class Person : IPerson
{
    private string _firstname = null;
    private string _lastname = null;
    private DateTime _birthday;

    public string Firstname
    {
        get { return this._firstname; }
        set { this._firstname = value; }
    }

    public string Lastname
    {
        get { return this._lastname; }
        set { this._lastname = value; }
    }

    public DateTime Birthday
    {
        get { return this._birthday; }
        internal set { this._birthday = value; }
    }
}
```

Der internal Zugriffsmodifizierer sorgt dafür, dass der Setter in derselben Assembly aufgerufen werden kann. Dies ist in vielen Fällen sehr hilfreich.

2.2.68.Generische Methoden und deren Aufruf

Einfach auf der Zunge zergehen lassen oder in eine Consolen-Anwendung kopieren und austesten.

```
class Program
{
    static void Function<T>(int x, T y)
    {
        Console.WriteLine("Function 1");
    }

    static void Function<T>(T x, long y)
    {
        Console.WriteLine("Function 2");
    }

    static void Main()
    {
        Function<int>(5, 324);
        Function<byte>(5, 324);
        Function(5, 324);
        Console.WriteLine();
    }
}
```

2.2.69.Variable Anzahl an Methoden-Parametern

Wer eine variable Anzahl an Parametern in einer Methode benötigt, der hat vermutlich bereits das Schlüsselwort params entdeckt. Wer nicht, dem sei es hiermit erklärt:

Das Schlüsselwort params ermöglicht es, einer Methode eine beliebige Anzahl an Parametern zu übergeben. Im Gegensatz zu einem Array können die einzelnen Werte mit Hilfe eines Komas getrennt werden, so wie es beispielsweise auch bei einem String.Format funktioniert.

Im folgenden Beispiel wird eine generische Liste durch eine Methode AddMultipleStringsToList erweitert, welche diese Funktionalität bietet:

```
public class MultipleStringList : List<string>
{
public void AddMultipleStringsToList(params string[] values)
{
foreach (string s in values)
```

```
this.Add(s);
}
}
```

Zusätzlich zu den Methoden `Add` und `AddRange` ist nun die neue Methode `AddMultipleStringsToList` verfügbar, die folgendermaßen verwendet werden kann:

```
MultipleStringList mStringList = new MultipleStringList();
mStringList.AddMultipleStringsToList("test1", "test2", "test3", "test4");

foreach (string s in mStringList)
Console.WriteLine(s);
```

2.2.70.Schlüsselwort sealed

Von einer Klasse die mit dem `sealed`-Schlüsselwort gekennzeichnet ist, kann nicht vererbt werden (sealed = versiegelt). Dies bedeutet, dass Erweiterungen nicht gewünscht sind bzw. eine weitere Abstraktion ohnehin sinnlos wäre. Aufrufe an versiegelte Klassenmember können durch Laufzeitoptimierungen beschleunigt werden.

Beispiel

```
public sealed class TestClass
{
private int _a = 0;
private int _b = 0;

public int A
{
get { return this._a; }
set { this._a = value; }
}

public int B
{
get { return this._b; }
set { this._b = value; }
}
}
```

Ergänzungen

Strukturen (*structs*) sind per Definiton `sealed` und können daher nicht vererbt werden.
Eine Verwendung von `sealed` gemeinsam mit `abstract` ist nicht erlaubt, da man bei einer abstrakten Klasse davon ausgehen kann, dass die konkrete Implementierung nicht existiert.

2.2.71.Unregelmäßigkeiten beim Implementieren einer abstrakten Klasse

Beim Ableiten von einer abstrakten Klasse können bekanntlich mittels STRG-SHIFT-F10 die zu implementierenden Methoden automatisch eingefügt werden. Dies sieht dann beispielsweise so aus:

```
public override void BeginTransaction()
{
throw new Exception("The method or operation is not implemented.");
}
```

Allerdings frage ich mich, warum keine NotImplementedException verwendet wird, welche bereits seit .NET 1.0 im Framework vorhanden ist.

2.2.72.C#: Arbeit mit der Registry

Die Registry wird noch immer häufig verwendet, um bestimmte Daten zu hinterlegen. .NET bietet hierfür die Klasse RegistryKey.

Hier ein Beispielcode wie Daten in die Registry geschrieben werden können:

```
RegistryKey rk = Registry.CurrentUser;
RegistryKey rkSoftware =
  rk.OpenSubKey("Software", true);
if (rkSoftware != null)
{
    RegistryKey rkCompany =
      rkSoftware.OpenSubKey("Norbert Eder",true);
    if (rkCompany == null)
    {
        rkCompany = rkSoftware.CreateSubKey("Norbert Eder");
    }
    RegistryKey rkInstaller =
      rkCompany.OpenSubKey("MySoftware", true);
    if (rkInstaller == null)
        rkInstaller = rkCompany.CreateSubKey("MySoftware");
    rkInstaller.SetValue("value1", "test1");
    rkInstaller.SetValue("value2", "test2");
    rkInstaller.Close();
    rkCompany.Close();
    rkSoftware.Close();
}
rk.Close();
```

Natürlich müssen die Werte auch wieder ausgelesen werden. Dies passiert auf folgende Art und Weise:

```
RegistryKey rk =
    Registry.CurrentUser.OpenSubKey
      (@"Software\Norbert Eder\MySoftware");
if (rk != null)
{
    string value1 = (string)rk.GetValue("value1");
    string value2 = (string)rk.GetValue("value2");
```

```
    rk.Close();
}
```

Damit sollte es nun einfach möglich sein, Werte in die Registry zu schreiben und daraus auszulesen. Eventuell empfiehlt es sich, hier noch genauer im MSDN nachzulesen, um zusätzliche Informationen zu erhalten.

2.2.73.Exception Handling und Security

Zum Exception Handling habe ich bereits hier, hier und hier berichtet. Was aber bis dato gefehlt hat, war eine Aussage zum Thema Sicherheit bei der Behandlung von Ausnahmen.

Was also hat das Behandeln von Ausnahmen mit Sicherheit zu tun?

In den meisten Fällen wird bei einer Exception der Inhalt der Eigenschaft **Message** zurückgegeben und in der Hauptanwendung (egal ob Windows Forms, Web, WPF) zur Anzeige gebracht. Dadurch werden jedoch in manchen Fällen Daten zum Vorschein gebracht, die besser im Verborgenen bleiben sollten. Nehmen wir das Beispiel Webanwendung. Gehen wir weiters davon aus, dass diese eine Verbindung zu einer Datenbank benötigt. Nun wird hier im System ein ConnectionString hinterlegt (natürlich gilt es auch diesen abzusichern).

Nun kann folgendes Problem auftreten:

Die Anmeldung auf den Datenbank-Server schlägt fehl. Daraufhin wird eine Exception geworfen, welche dann im User-Interface angezeigt wird (entsweder per eigener Fehlerseite oder überhaupt als Exception). Aus dem Message-Text ist nun ersichtlich, dass die Anmeldung scheitterte und mit welchem User die Anmeldung versucht wurde.

Ein potentieller Angreifer hat nun ein leichteres Spiel, da er einen User für die Datenbank definitiv kennt.

Dies ist nur ein einfaches Beispiel. Aus diesem Grunde sollten Exception-Messages niemals direkt an den User weitergegeben werden. Folgende Vorgehensweise ist hier empfohlen:

1. Jede Exception abfangen
2. Exceptions in eine Log-Datei loggen (Bei Webanwendungen sollte die Log-Datei in ein Verzeichnis geschrieben werden, welches nicht über das Web zugänglich ist)
3. Fehlertexte, die an den User gehen sollten unbedingt zuvor angepasst werden. D.h. ein eigener Wortlaut muss deklariert werden.
4. Schließlich bleibt noch zu erwähnen, dass dem User nicht jeder Fehler sichtbar gemacht werden muss. Mit vielen Fehlern kann der Unser ohnehin nichts anfangen und sie verwirren ihn nur. Ergo immer überlegen, ob die Benachrichtigung im speziellen Fall sinnvoll ist oder nicht.

2.2.74.C# 3.0: Anonyme Typen

Nachdem ich bereits über Anonymen Delegates (auch unter .NET 2.0 verfügbar) als auch dem unter C# 3.0 neuen Keyword var geschrieben habe, möchte ich ein wenig über **Anonyme Typen** schreiben.

Anonyme Typen sind ähnlich zu anonymen Methoden (dazu kommen wir dann auch in einem der nächsten Blogbeiträge).Dies bedeutet, dass anonyme Typen inline gebildet werden können. Um dies zu verdeutlichen gleich ein kleines Beispiel:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace AnonymousTypesDemo
{
    class Program
    {
        static void Main(string[] args)
        {
            var myType = new { Firstname = "Norbert", Lastname = "Eder" };

            myType.
        }
    }
}
```

Equals
Firstname
GetHashCode
GetType
Lastname
ToString

string 'a.Firstname

Anonymous Types:
'a is new { string Firstname, string Lastname }

Abbildung 7: Anonyme Typen und IntelliSense

Wie zu sehen ist, ist hierfür das Schlüsselwort **var** zu verwenden. Die Definition erfolgt inline und definiert welche Eigenschaften der neue Typ hat. Selbst IntelliSense kennt die neuen Eigenschaften.

Anonyme Typen kennzeichnen sich dadurch aus, dass sie keinen zugewiesenen Namen besitzen (dies ist auch in der Typen-Beschreibung - siehe IntelliSense - ersichtlich).

Werden im gleichen Scope zwei anonyme Typen definiert, welche die gleiche Signatur besitzen, kann der erste Type den zweiten einfach übernehmen, wie nachfolgend zu sehen:

```
var person = new { Firstname = "Norbert", Lastname = "Eder", CarsCount = 1};

var anotherPerson = new { Firstname = "Der", Lastname = "Tester", CarsCount = 2 };

person = anotherPerson;
```

Abbildung 8: Kompatible anonyme Typen

Wozu werden anonyme Typen nun gebraucht?

Zum einen wurden anonyme Typen natürlich für LINQ eingeführt. Diese werden dort

häufig verwendet und bilden daher eine Notwendigkeit. Aber auch für andere Zwecke können sie verwendet werden.

Grundsätzlich bieten sich anonyme Type dann an, wenn ein Typ kurzfristig (für die Ausgabe etc.) benötigt wird, aber nicht für die gesamte Anwendung zur Verfügung stehen muss. In diesem Fall muss also keine eigene Klasse erstellt werden, welche schlussendlich nur an einer einzigen Stelle verwendet werden würde. Wann anonyme Typen einzusenden sind bleibt jedem Entwickler selbst überlassen. Unter dem obigen Gesichtspunkt sollte aber durchaus klar sein, wann eine Verwendung sinnvoll ist und wann weniger.

2.2.75.Anonymous Delegates

Dem einen oder anderen mögen anonyme Methoden (oder beispielsweise anonyme Typen wie sie mit C# 3.0 kommen) bekannt sein. In diesem Beitrag möchte ich ein wenig über anonyme Delegates schreiben und wozu diese benötigt werden (sind ohnehin sehr ähnlich zu anonymen Methoden).

Zuerst gleich das Beispiel, anhand dessen die Erklärungen vorgenommen werden:

```
using System;
using System.Collections.Generic;
using System.Text;
namespace AnonymousDelegatesSample
{
    public class Book
    {
        public string Title;
        public decimal Price;
    }
    public class Person
    {
        public string Firstname;
        public string Lastname;
        public List<Book> BookList;
        public decimal Discount;
        public DiscountCalculator CalculationMethod;
    }
    public delegate decimal DiscountCalculator(decimal price);
    public class Example
    {
        private decimal CalculateDiscount(decimal price)
        {
            return price * new decimal(0.05);
        }
        public void ShowExample()
        {
            DiscountCalculator default_calc =
                new DiscountCalculator(CalculateDiscount);
            DiscountCalculator specialcustomer_calc =
                delegate(decimal price)
                {
                    return price * new decimal(0.09);
                };
            Book book1 = new Book();
            book1.Title = "ASP.NET for dummies";
            book1.Price = new decimal(35.99);
            Book Book2 = new Book();
            Book2.Title = "C# for dummies";
            Book2.Price = new decimal(49.99);
            List<Book> bookList = new List<Book>();
```

```
            bookList.Add(book1);
            bookList.Add(Book2);
            List<Person> personList = new List<Person>();
            Person person1 = new Person();
            person1.Firstname = "Norbert";
            person1.Lastname = "Eder";
            person1.BookList = bookList;
            person1.CalculationMethod = specialcustomer_calc;
            personList.Add(person1);
            Person person2 = new Person();
            person2.Firstname = "Alf";
            person2.Lastname = "Dudldei";
            person2.BookList = bookList;
            person2.CalculationMethod = default_calc;
            personList.Add(person2);
            foreach (Person p in personList)
            {
                decimal price = 0;
                foreach (Book b in p.BookList)
                    price += b.Price;
                p.Discount = p.CalculationMethod(price);
                Console.WriteLine(
                    String.Format("{0} {1}:t {2} discount",
                        p.Firstname,
                        p.Lastname,
                        p.Discount.ToString()));
            }
            Console.WriteLine("Press any key ...");
            Console.ReadKey();
        }
    }
}
```

Zuerst ein kurze Erklärung wozu dieses Beispiel dient: Grundsätzlich gibt es zwei Klassen. Zum einen die **Person** und zum anderen das **Book**. Wir gehen also davon aus, dass eine Person Bücher kauft und diese in einer Collection aufbewahrt werden. Nun soll (je nach Kundenstatus) ein Rabatt berechnet werden.

Zur Erklärung: Dies kann natürlich auch auf eine andere - vermutlich einfachere Art und Weise implementiert werden. Dieses Beispiel dient lediglich der Veranschaulichung von anonymen Delegates.

Grundsätzlich ist in diesem Beispiel ein üblicher Delegate definiert:

```
public delegate decimal DiscountCalculator(decimal price);
```

Dieser wird im Standardfall auch verwendet, um eine Methode aufzurufen, die für die Rabattberechnung zuständig ist.

```
DiscountCalculator default_calc =
    new DiscountCalculator(CalculateDiscount);
```

Die aufgerufene Methode sieht dann so aus:

```
DiscountCalculator default_calc =
    new DiscountCalculatprivate decimal CalculateDiscount(decimal price)
{
    return price * new decimal(0.05);
}or(CalculateDiscount);
```

Nun kann es jedoch vorkommen, dass eine spezielle Berechnung durchgeführt werden muss, diese wird nur in einem ganz bestimmten Fall angewandt und bedarf

daher keiner eigenen Methoden-Implementierung (in diesem Beispiel vielleicht weniger sinnvoll, aber generell treten bei der Verwendung von Delegates immer wieder Fälle auf, in denen sich eine eigene Methode nicht rentiert). Hier kommen nun anonyme Delegates ins Spiel. Die aufgerufene Methode ist hierbei ansich nicht vorhanden (daher anonym) und wird erst bei der Erstellung des Delegate definiert. Wie hier:

```
DiscountCalculator specialcustomer_calc =
    delegate(decimal price)
    {
        return price * new decimal(0.09);
    };
```

Damit wird im Grunde erreicht, dass die eigentliche Berechnung durch eine neue ersetzt wird. Eine entsprechende Methode muss hierzu nicht vorhanden sein.

Einsatzgebiete hierfür finden sich immer wieder, daher grundsätzlich nicht schlecht, wenn man sie kennt und auch anwenden kann. Im Falle des Falles muss der Entwickler ohnehin selbst entscheiden ob dies sinnvoll ist oder nicht. Wird die Berechnung anderweitig verwendet ist es sicherlich sinnvoll, diese in eine eigene Methode zu kapseln. Andernfalls kann durchaus ein anonymer Delegate verwendet werden.

2.2.76.C# 3.0: Keyword var

Nein, `var` steht hier nicht in nostalgischer Verbindung zu Visual Basic. Vielmehr handelt es sich – streng genommen – auch nicht wirklich um ein Schlüsselwort (Keyword). Schlüsselwörter dürfen ja nicht in Klassen, Strukturen etc. verwendet werden, `var` hingegen sehr wohl. Aber unabhängig dieser Tatsache, was verbirgt sich dahinter?

Beginnen wir mit einem kleinen Beispiel:

```
var htTemp = new Hashtable();
```

In diesem Fall versucht der Compiler auf den Typ von strList zu schließen, was auch kein Problem darstellt, da dieser ohnehin durch new Hashtable() definiert wird. Es entfällt also die lange und natürlich auch unschöne und redudante Darstellung á la:

```
Hashtable htTemp = new Hashtable();
```

Manche mögen nun der Meinung sein, dass sich C# dadurch von der starken Typisierung verabschiedet. Dem ist jedoch nicht so. Wurde die Variable deklariert, wird auf ihren Typ geschlossen und zugewiesen. Der Typ kann an einer späteren Stelle nicht mehr verändert werden (wie dies beispielsweise unter VB.NET möglich ist).

Ebenfalls sei zu betonen, dass var auch nur bei der Initialisierung verwendet werden kann.

2.2.77.C#: Eintrag in das EventLog hinzufügen

Das EventLog stellt eine gute Basis dar, Informationen, oder aufgetretene Probleme abzulegen. Hier ein kleines Beispiel, wie dies funktioniert:

```
if (!EventLog.SourceExists("TestCategory"))
    EventLog.CreateEventSource("TestCategory", "TestLog");
EventLog evtLog = new EventLog();
evtLog.Source = "TestCategory";
evtLog.WriteEntry("This is a test", EventLogEntryType.Information);
```

Zuerst wird lediglich überprüft, ob eine bestimmte Kategorie im EventLog bereits vorhanden ist. Wenn nicht erfolgt eine Anlage.

Anschließend wird ein Objekt vom Typ **EventLog** erstellt, als **Source** die soeben angelegte Kategorie zugewiesen. Mittels **WriteEntry** kann nun ein Eintrag erstellt werden. **EventLogEntryType** legt den Typ des zu schreibenden Eintrages fest.

2.2.78..NET: Directory.Move und unterschiedliche Volumes

Mit Directory.Move ist es sehr einfach, komplette Verzeichnisse zu verschieben. Ein Problem stellt sich lediglich dann ein, wenn Verzeichnisse zwischen unterschiedlichen Volumes verschoben werden sollen. Dies wird durch Directory.Move nicht abgedeckt. Stattdessen muss auf eine eigene Implementierung zurück gegriffen werden.

Mit Hilfe einer Rekursion kann das dann in etwa so aussehen:

```
public void MoveDirectory(
    string strSourceDir,
    string strDestDir,
    bool bDelSource)
{
    if (Directory.Exists(strSourceDir))
    {
        if (Directory.GetDirectoryRoot(strSourceDir) ==
            Directory.GetDirectoryRoot(strDestDir))
        {
            Directory.Move(strSourceDir, strDestDir);
        }
        else
        {
            try
            {
                CopyDirectory(new DirectoryInfo(strSourceDir),
                    new DirectoryInfo(strDestDir));
                if (bDelSource)
                    Directory.Delete(strSourceDir, true);
            }
            catch (Exception subEx)
            {
                throw subEx;
            }
        }
    }
}
private void CopyDirectory(
```

```
    DirectoryInfo diSourceDir,
    DirectoryInfo diDestDir)
{
    if (!diDestDir.Exists)
        diDestDir.Create();
    FileInfo[] fiSrcFiles = diSourceDir.GetFiles();
    foreach (FileInfo fiSrcFile in fiSrcFiles)
        fiSrcFile.CopyTo(Path.Combine(diDestDir.FullName, fiSrcFile.Name));
    DirectoryInfo[] diSrcDirectories = diSourceDir.GetDirectories();

    foreach (DirectoryInfo diSrcDirectory in diSrcDirectories)
        CopyDirectory(
            diSrcDirectory,
            new DirectoryInfo(
                Path.Combine(diDestDir.FullName, diSrcDirectory.Name)));
}
```

Was ich an `Directory.Move` allerdings nicht ganz verstehe ist, dass eben Verzeichnisse nicht auf unterschiedliche Volumes verschoben werden können. File.Move kann dies, warum also `Directory.Move` nicht.

2.2.79.C#: Entwicklung einer Notiz-Anwendung anhand eines Beispiels

Nachdem ich heute zufällig darüber gestolpert bin, möchte ich diese Artikel-Serie natürlich nicht vorenthalten. Worum geht es konkret:

StickOut ist eine Desktopanwendung für Kurznotizen mit Unterstützung für mehrere Benutzer und Integration in Outlook. Es handelt sich um eine Windows Forms-Anwendung für .NET Framework 2.0, bei der die Kommunikation mit anderen StickOut-Benutzern sowie der Austausch von Kurznotizen über .NET Remoting stattfindet. Der neue IPC-Kanal von .NET Framework 2.0 wird zur Kommunikation zwischen Microsoft Outlook und dem StickOut-Prozess verwendet. Mit dem Outlook-Add-In können Sie von beliebigen Outlook-Objekten Kurznotizen auf dem Desktop erstellen, einschließlich E-Mails, Notizen, Aufgaben, Terminen usw. Es wurde darauf geachtet, den Speicherbedarf einer .NET-Anwendungen maßgeblich zu verringern und einen zügigen und reibungslosen Umgang für den Benutzer zu gewährleisten. Dieser mehrteilige Artikel enthält einen Bericht über die Entstehung, beginnend mit meinem ersten Tag der Anwendungsplanung, hin zu allen nachfolgenden Entwurfs-, Entwicklungs-, Test- und Bereitstellungsphasen. Er zeigt die Fortentwicklung der Anwendung bis hin zu den Komplikationen, die bei den einzelnen Schritten aufgetreten sind. Sie erfahren vieles über .NET-Tricks, Bereitstellungs- und Versionskontrollprobleme, Visual Studio-Tricks und weitere nicht direkt auf die Entwicklung bezogene Tricks, die für Ihre tägliche Entwicklungsarbeit nützlich sein können.

Wer daran Interesse hat, hier die Links zu den einzelnen Teilen dieser Serie:

StickOut: Entstehung einer Kurznotiz-Anwendung in .NET 2.0, Teil 1

StickOut: Entstehung einer Kurznotiz-Anwendung in .NET 2.0, Teil 2

StickOut: Entstehung einer Kurznotiz-Anwendung in .NET 2.0, Teil 3

2.2.80.C#: Schnell und generisch Objekte erstellen

Oft müssen Objekte dynamisch bzw. generisch erzeugt werden. In den meisten Fällen wird zu **`Activator.CreateInstance`** oder **`Activator.CreateInstance<>`** gegriffen. Die Erzeugung über den **`Activator`** ist jedoch nicht die schnellste. Einen wesentlich performantereren Ansatz liefern uns die **`FormatterServices`**. Hier eine Objekt-Erzeugungs-Klasse, welche beide Varianten implementiert:

```
public class ObjectGenerator
{
    public T GenerateObject<T>()
    {
        return (T)FormatterServices.GetUninitializedObject(typeof(T));
    }
    public T ActivateObject<T>()
    {
        return (T)Activator.CreateInstance<T>();
    }
}
```

Erzeugen wir und für jede Variante in einer Schleife 10.000 Objekte, sehen wir eine kleine Überraschung. Zuerst der Code:

```
ObjectGenerator og = new ObjectGenerator();
Stopwatch sw = new Stopwatch();
Console.Write("Generate Object [Formatter]: ");
sw.Start();
for (int i = 0; i < 10000; i++)
{
    TestClass tc = og.GenerateObject<TestClass>();
}
sw.Stop();
Console.WriteLine(sw.ElapsedMilliseconds);
sw.Reset();
Console.Write("Generate Object [Activator]: ");
sw.Start();
for (int i = 0; i < 10000; i++)
{
    TestClass tc = og.ActivateObject<TestClass>();
}
sw.Stop();
Console.WriteLine(sw.ElapsedMilliseconds);
Console.ReadKey();
```

Nun die Überraschung:

```
Generate Object [Formatter]: 3
Generate Object [Activator]: 21
```

Und jetzt kommt **der eigentliche Clou**: Mit Hilfe von **`FormatterServices.GetUninitializedObject`** ist es möglich Objekte von Klassen mit privaten Konstruktoren (kein öffentlicher Konstruktor vorhanden!!) zu erstellen. Hier eine Beispielklasse:

```
public class TestClass
{
```

```
    private TestClass() { }
    public string GetMessage()
    {
        return "This is a message";
    }
}
```

Diese kann mit **`GetUninitializedObject`** instantiiert werden, wogegen `CreateInstance` eine `MissingMethodException` wirft und uns darauf aufmerksam macht:

```
Für dieses Objekt wurde kein parameterloser Konstruktor definiert.
```

Einsatzmöglichkeiten dafür werden sich sicherlich für manche finden.

2.2.81. C#: BinaryFormatter, SerializationException und dynamisch geladene Assemblies

Zwei Solutions. In der ersten Solution befindet sich eine Windows Forms Anwendung. Diese referenziert eine Assembly aus der zweiten Solution. In der zweiten Solution befindet sich eine Konsolen-Anwendung, welche eben erwähnte Assembly per Assembly.GetAssembly lädt. Zu erwähnen sei ebenfalls, dass folgende Klassen enthalten sind:

- Datenklassen (diese werden serialisiert)
- Commands (siehe Command Pattern)

Nun gut. In der Windows Forms Anwendung werden die einzelnen Commands konfiguriert, und per BinaryFormatter serialisiert. Dabei ergibt sich jedoch ein kleines Problem:

Da die Konfigurationen in der Windows Forms Anwendung serialisiert wurden (hierzu muss angemerkt werden, dass sich die Klasse, welche die Konfigurationen liest und auch wieder schreibt nicht in der besagten Assembly befindet) und in der zweiten Solution (mit der Konsolen-Anwendung) dynamisch geladen werden sollte, wurde eine SerializationException geworfen. Beim Binary-Formatter wird unter anderem die genaue Assembly-Bezeichnung, Versionsnummer etc. hinterlegt, die zwar in diesem Fall zusammenpassen würde, aber dennoch konnten die entsprechenden Typen nicht gefunden werden (Referenz war gesetzt, Assembly wurde in die aktuelle AppDomain geladen).

Ein Ausweg ist hierbei das Setzen der Eigenschaft **Binder** des BinaryFormatters. Durch eine Ableitung der Klasse SerializationBinder kann das Verhalten, wie Typen geladen bzw. gefunden werden, selbst beeinflusst werden. In meinem Fall habe ich folgendes fabriziert, was dann auch wunderbar funktionierte:

```
public class Binder : SerializationBinder
{
    public override Type
```

```
        BindToType(string assemblyName, string typeName)
    {
        Type type = null;
        string shortAssem = assemblyName.Split(',')[0];
        Assembly[] assemblies =
            AppDomain.CurrentDomain.GetAssemblies();
        foreach (Assembly assem in assemblies)
        {
            if (shortAssem == assem.FullName.Split(',')[0])
            {
                type = assem.GetType(typeName);
                break;
            }
        }
        return type;
    }
}
```

Wer weiterführende Informationen diesbezüglich benötigt wird entweder via Google fündig, oder stellt seine Frage einfach als Kommentar. Ich versuche diese dann zu beantworten.

2.2.82.C#: Rahmenfarbe beim Panel ändern

Durch das Setzen der Eigenschaft **BorderStyle** auf **FixedSingle** erhält das Panel einen 2D-Rahmen, der fix auf Schwarz eingestellt ist. In manchen Fällen soll nun eine andere Rahmenfarbe her. Dazu muss von Panel abgeleitet und der Handler **OnPaint** überschrieben werden. Hier ein kleines Beispiel, bei dem sowohl die Rahmenfarbe, als auch die Rahmenstärke bequem über Eigenschaften eingestellt werden kann.

```
public partial class PanelEx : Panel
{
    [DllImport("user32.dll")]
    private static extern IntPtr GetWindowDC(IntPtr hwnd);
    [DllImport("user32.dll")]
    private static extern int ReleaseDC(IntPtr hwnd, IntPtr hdc);
    private Color _borderColor = Color.Black;
    private int _borderWidth = 1;
    public int BorderWidth
    {
        get { return _borderWidth; }
        set { _borderWidth = value; }
    }
    public Color BorderColor
    {
        get { return _borderColor; }
        set { _borderColor = value; }
    }
    public PanelEx()
    {
        this.InitializeComponent();
        SetStyle
            (ControlStyles.DoubleBuffer,
            true);
        SetStyle
            (ControlStyles.AllPaintingInWmPaint,
            false);
        SetStyle
            (ControlStyles.ResizeRedraw,
            true);
        SetStyle
```

```
            (ControlStyles.UserPaint,
            true);
        SetStyle
            (ControlStyles.SupportsTransparentBackColor,
            true);
    }
    protected override void OnPaint(PaintEventArgs e)
    {
        base.OnPaint(e);
        if (this.BorderStyle == BorderStyle.FixedSingle)
        {
            IntPtr hDC = GetWindowDC(this.Handle);
            Graphics g = Graphics.FromHdc(hDC);
            ControlPaint.DrawBorder(
                g,
                new Rectangle(0, 0, this.Width, this.Height),
                _borderColor,
                _borderWidth,
                ButtonBorderStyle.Solid,
                _borderColor,
                _borderWidth,
                ButtonBorderStyle.Solid,
                _borderColor,
                _borderWidth,
                ButtonBorderStyle.Solid,
                _borderColor,
                _borderWidth,
                ButtonBorderStyle.Solid);
            g.Dispose();
            ReleaseDC(Handle, hDC);
        }
    }
}
```

Zusätzliche Erweiterungen sind natürlich denkbar und können gerne von jedem vorgenommen werden.

2.2.83.C#: Feststellen ob eine Assembly signiert wurde (strong named)

Wurde eine Anwendung signiert, dann müssen ebenfalls alle referenzierten Assemblies signiert, also strong named, sein. Aussnahme: Assemblies werden zur Laufzeit geladen. In diesem Fall muss diese nicht signiert sein. Wie kann man nun aber feststellen, ob eine Assembly signiert wurde oder nicht? Hier ein kurzes Code-Beispiel:

```
Assembly asm = Assembly.GetAssembly( typeof( TheType) );
if (asm != null)
{
    AssemblyName assembly = asm.GetName();
    byte[] key = assembly.GetPublicKey();
    bool isSigned = key.Length > 0;
    Console.WriteLine("Is signed: {0}", isSigned);
}
```

Dies kann sehr nützlich sein, wenn beispielsweise Plugins geladen werden, die jedoch signiert sein sollten. Hier gilt es jedoch aufzupassen: Wird eine Assembly in die aktuelle Anwendungsdomäne geladen, kann die Assembly nicht mehr explizit entladen werden. Es empfiehlt sich daher, die Assembly über einen Remote-Proxy in

eine eigene Anwendungsdomäne zu laden, auf Signierung hin zu überprüfen und bei Bedarf (also falls nicht signiert), die neue Domäne wieder zu entladen.

2.2.84.EXIF Informationen mit C# und Boardmitteln auslesen

Neulich musste ich aus JPEG-Dateien EXIF-Daten auslesen. Hierzu gibt es unterschiedliche Möglichkeit. So kann man sich auf externe Bibliotheken verlassen, welche diese Funktionalität anbieten, oder es wird mit Hilfe der .NET 2.0+ Boardmittel umgesetzt. Ich habe mich für die zweite Variante entschieden, die zwar einen entscheidenden Nachteil mit sich bringt, der in meinem Fall jedoch nicht schlagend wurde: Performance.

So musste ich nur das Erstellungsdatum des Bildes auslesen (hier konnte ich nicht das Dateidatum heranziehen, da das Bild eventuell nachbearbeitet wurde). Das kann mit folgendem Code realisiert werden:

```
public class ExifReader
{
    public string ReadExifDate(Image image)
    {
        PropertyItem[] items = image.PropertyItems;
        foreach (PropertyItem pi in items)
        {
            if (pi.Id == 306)
            {
                string val =
                    System.Text.Encoding.Default.GetString
                        (pi.Value);
                return val;
            }
        }
        return "n/a";
    }
}
```

Die Eigenschaft PropertyItem.Id repräsentiert grundlegend einen HEX-Wert, welcher angibt, welche Informationen sich im PropertyItem befinden. Dadurch könnte dieses Beispiel wesentlich erweitert werden. Sämtliche IDs sind unter dem angegebenen Link ersichtlich.

Der angegebene Nachteil besteht nun darin, dass für diese Variante, jedes Image (beispielsweise über Image.FromFile) geladen werden muss. Dies kostet jede Menge Performance. Deshalb sollte bei der Abarbeitung von großen und vor allem vielen Bildern diese Variante nicht herangezogen werden.

EXIF-Bibliotheken sollten durchaus einfach zu finden sein. Hier dennoch eine mögliche Library, die jedoch nicht auf Geschwindigkeit hin von mir getestet wurde:

EXIFextractor

2.2.85.C#: Der as-Operator

Einführung as-Operator

Der as-Operator ist ähnlich einer Cast-Operation. Im Gegensatz zu einem Cast liefert as jedoch keine Ausnahme (Exception), sondern gibt null zurück. Ein Beispiel:

```
object o = new MyDemoClass();
MyDemoClass mdc = o as MyDemoClass;
```

Die Cast-Variante würde wie folgt aussehen:

```
object o = new MyDemoClass();
MyDemoClass mdc = (MyDemoClass)o;
```

Ein Äquivalent zum as-Operator könnte so formuliert werden:

```
object o = new MyDemoClass();
MyDemoClass demoClass =
   (o is MyDemoClass) ? (MyDemoClass)o : null;
```

Für benutzer-definierte Konvertierungen muss ein Cast verwendet werden, da dies durch den as-Operator nicht abgebildet wird (nur Referenzen und Boxing).

Performance

Bezüglich der Performance gibt es einen kleinen Unterschied, der sich allerdings erst bei einer hohen Anzahl an Durchläufen auswirkt. Ein Test mit 1 Milliarde Iterationen zeigte einen deutlichen Unterschied, sollte jedoch in der Praxis in dieser Form nicht allzu oft vorkommen. Hier die Testanwendung:

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Diagnostics;
namespace AsTest
{
    class Program
    {
        static void Main(string[] args)
        {
            Stopwatch sw = new Stopwatch();
            object o = new MyDemoClass();
            sw.Start();
            for (int i = 0; i < 1000000000; i++)
            {
                MyDemoClass mdc = o as MyDemoClass;
            }
            sw.Stop();
            Console.WriteLine("as: " +
                sw.ElapsedMilliseconds);

            sw.Reset();
            sw.Start();
            for (int i = 0; i < 1000000000; i++)
            {
```

```
                MyDemoClass mdc = (MyDemoClass)o;
            }
            sw.Stop();
            Console.WriteLine("cast: " +
                sw.ElapsedMilliseconds);
            Console.ReadKey();
        }
    }
    public class MyDemoClass
    {
    }
}
```

Output

```
as: 6135
cast: 5583
```

Fazit

In vielen Fällen wäre der as-Operator einem Cast vorzuziehen, da hier einer eventuellen Exception vorgebeugt werden kann (im Falle eines Casts müsste davor via is-Operator überprüft werden, ob es sich um den gewünschten Typ handelt). Auf jeden Fall muss das Objekt anschließend auf null überprüft werden.

2.2.86.[Enterprise Library] NLog Trace Listener im Eigenbau

Eventuell hat sich der werte Leser die Enterprise Library [1] bereits einmal genauer angesehen. Wenn nicht, dann möchte ich zuerst ein paar einführende Worte loswerden.

Die Enterprise Library besteht aus sogenannten Blocks. Jeder Block besitzt eine eindeutige Aufgabe. Beispielsweise gibt es den Logging und den Exception Handling Block. Der Vorteil besteht darin, dass es mit Hilfe der Blocks das Verhalten über die Konfiguration (App.config, Web.config, ...) gesteuert werden kann. Beispiel:

Bei Exceptions stellt sich die Frage, wie denn diese im Spezialfall zu behandeln sind. Sind sie für den User sichtbar, sollen sie geloggt werden, oder gar der schlimmste Fall: soll sie nicht behandelt werden? Die geworfenen Exceptions werden an den Exception Handling Block übergeben, welcher durch eine Konfigurationsdatei entsprechend eingerichtet wird. Im Nachhinein kann nun festgelegt werden, was mit bestimmten Exceptions passieren soll. Ebenso verhält es sich mit den anderen Blocks.

Beim Logging verhält es sich gleich. Durch die Konfiguration kann festgelegt werden, wohin geloggt werden soll. Hier stehen unterschiedlichste Möglichkeiten bereit: EventLog, Datenbank, Email, Flat File usw. In unserem Beispiel möchten wir allerdings alle geworfenen Exceptions via NLog aufzeichnen.

Sehen wir uns dazu allerdings das Grundgerüst an. Das Beispiel läuft als Konsolenanwendung und Instanziert eine Klasse **DoSomethingClass**, welche eine einzige Methode **ThrowAnException** besitzt:

```
using System;
using System.Collections.Generic;
using System.Text;
using Microsoft.Practices.EnterpriseLibrary.ExceptionHandling;
namespace ExceptionLoggingDemo
{
    public class DoSomethingClass
    {
        public void ThrowAnException()
        {
            try
            {
                int i = 0;
                i = 1 / i;
            }
            catch (Exception ex)
            {
                ExceptionPolicy.HandleException(ex, "DemoException");
                throw ex;
            }
        }
    }
}

Die Konsolen-Anwendung selbst sieht folgendermaßen aus:
using System;
using System.Collections.Generic;
using System.Text;
namespace ExceptionLoggingDemo
{
    class Program
    {
        static void Main(string[] args)
        {
            new DoSomethingClass().ThrowAnException();
            Console.WriteLine("Exception thrown");
            Console.ReadKey();
        }
    }
}
```

Nun binden wir die Enterprise Library ein bzw. die Teile davon, die wir tatsächlich benötigen. Dabei handelt es sich um folgende Libraries:

- Microsoft.Practices.EnterpriseLibrary.Common
- Microsoft.Practices.EnterpriseLibrary.ExceptionHandling
- Microsoft.Practices.EnterpriseLibrary.ExceptionHandling.Logging
- Microsoft.Practices.EnterpriseLibrary.Logging

Bevor wir nun daran gehen, die Enterprise Library zu konfigurieren, erstellen müssen wir uns genauer ansehen, wie der Ablauf aussehen soll.

Grundsätzlich binden wir den Exception Handling Block ein. Dieser fängt unsere Ausnahmen auf. Hier müssen wir eine neue **Exception Policy** hinzufügen, damit unsere Exception auch aufgefangen wird (hier sind unterschiedliche Policies möglich und können somit auch unterschiedlich behandelt werden). Dieser Policy hängen wir einen Logging Block um. Hierbei ist schön zu sehen, dass die einzelnen Blocks sehr gut miteinander zusammenarbeiten. Durch diese Definition legen wir fest, dass auftretende Exceptions mitgeloggt werden sollen. Nun muss der Logging Block entsprechend konfiguriert werden. Es muss also angegeben werden, wohin die

Daten geschrieben werden und wie diese formatiert werden sollen. Daraus ergeben sich zwei wichtige Bestandteile des Logging Blocks:

- Trace Listener
- Formatter

Der Trace Listener bekommt die zu schreibenden Daten übermittelt. Durch die Einbindung eines Formatters können diese Daten entsprechend formatiert werden, um besser lesbar bzw. einfacher zu verarbeiten sind. Als Formatter stehen ein TextFormatter und ein XmlFormatter zur Verfügung. Eigene Ableitungen sind an dieser Stelle natürlich möglich. Zudem wissen wir nun, an welcher Stelle wir ansetzen müssen: Wir benötigen einen **CustomTraceListener**.

Dazu wird ein eigenes Projekt erstellt, welches den benutzerdefinierten Listener enthalten wird. Nachfolgend der Sourcecode des Listeners:

```
[ConfigurationElementType(typeof(CustomTraceListenerData))]
public class NLogTraceListener : CustomTraceListener
{
    private LogLevel _logLevel = LogLevel.Info;
    public override void TraceData(
        TraceEventCache eventCache,
        string source,
        TraceEventType eventType,
        int id,
        object data)
    {
        if (data is LogEntry && this.Formatter != null)
        {
            switch (eventType)
            {
                case TraceEventType.Critical:
                    _logLevel = LogLevel.Fatal;
                    break;
                case TraceEventType.Error:
                    _logLevel = LogLevel.Error;
                    break;
                case TraceEventType.Information:
                    _logLevel = LogLevel.Info;
                    break;
                case TraceEventType.Warning:
                    _logLevel = LogLevel.Warn;
                    break;
                case TraceEventType.Verbose:
                    _logLevel = LogLevel.Debug;
                    break;
                default:
                    _logLevel = LogLevel.Info;
                    break;
            }
            this.WriteLine(this.Formatter.Format(data as LogEntry));
        }
        else
        {
            this.WriteLine(data.ToString());
        }
    }
    public override void Write(string message)
    {
        LogManager.GetCurrentClassLogger().Log(_logLevel, message);
    }
    public override void WriteLine(string message)
```

```
    {
        LogManager.GetCurrentClassLogger().Log(_logLevel, message);
    }
}
```

Insgesamt werden drei Methoden überschrieben (wobei ansich nur zwei Methoden tatsächlich benötigt werden):

- TraceData
- Write
- WriteLine

Wichtig sind zwei Dinge: Durch das Überschreiben der Methode TraceData kommen wir an den **TraceEventType**. Dieser gibt an, um welches Fehler-Level es sich handelt (Critical, Error, Warning, Info, ...). Dies benötigen wir, um es an NLog weitergeben zu können. Der Rest besteht ansich im Aufruf der Methode WriteLine unter Einbindung des zugewiesenen Formatters.

Schließlich muss noch eine **App.config** eingebunden werden. In dieser Konfigurations-Datei wird die Konfiguration der Enterprise-Library hinterlegt. Per rechter Maustaste auf die App.config kann die Enterprise Library Configuration gestartet werden, welche sich mittlerweile netterweise ins Visual Studio integriert. Zur Konfiguration müssen die Einträge für das Exception Handling und für das Logging hinzugefügt werden. Beim Logging verwenden wir unseren CustomTraceListener. Dazu ist eine Assembly anzugeben (unser separates Projekt). Zu beachten ist hier folgendes:

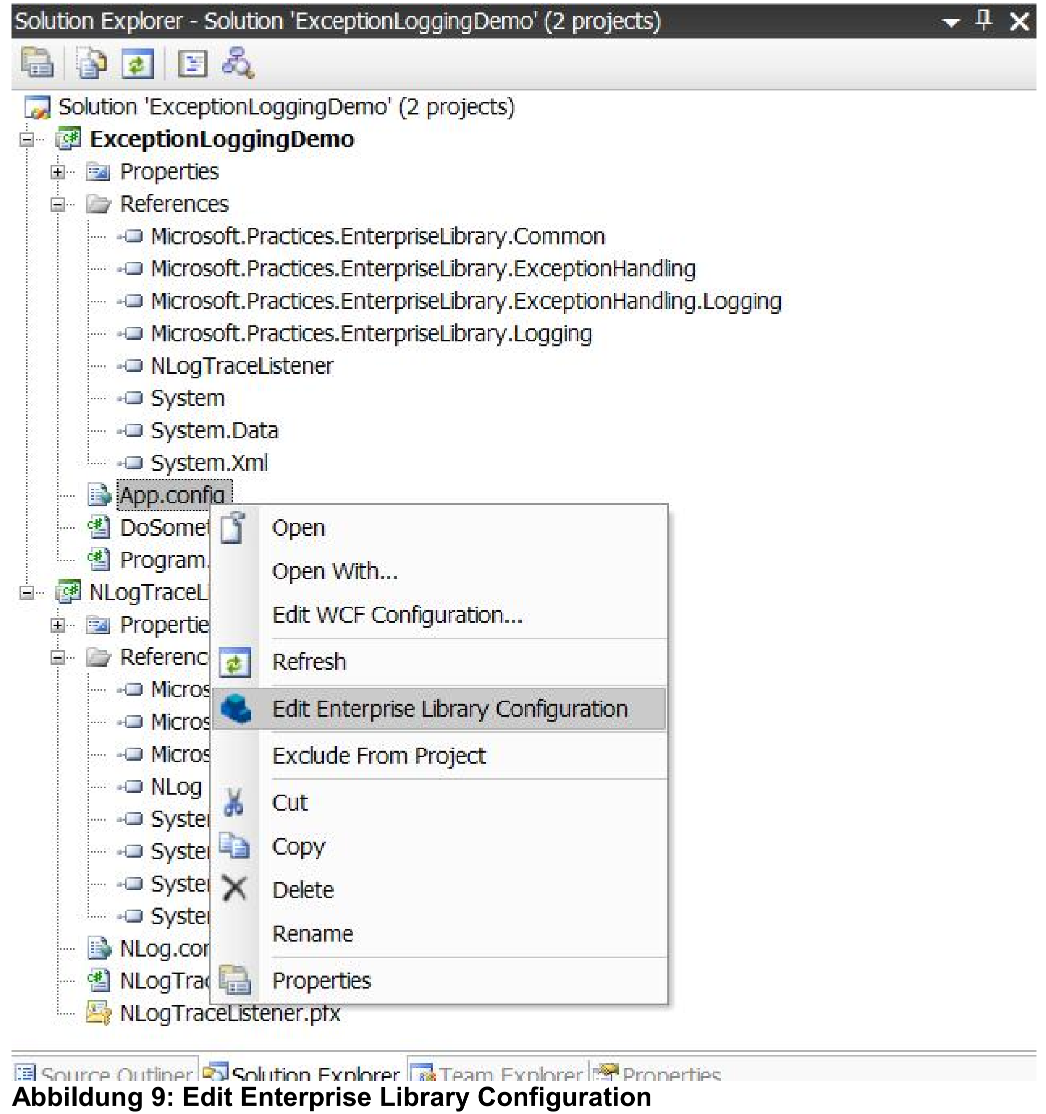


Abbildung 9: Edit Enterprise Library Configuration

Die Enterprise Library kann sowohl signiert als auch unsigniert eingebundern werden. Wird die signierte Variante verwendet, muss auch die eigene Assembly signiert sein. Zudem muss (im Falle eines TraceListeners) die Klasse von **CustomTraceListener** abgeleitet sein und folgendes Attribut gesetzt haben:

```
[ConfigurationElementType(typeof(CustomTraceListenerData))]
```

Damit ist alles getan, um unseren eigenen TraceListener laden zu können.

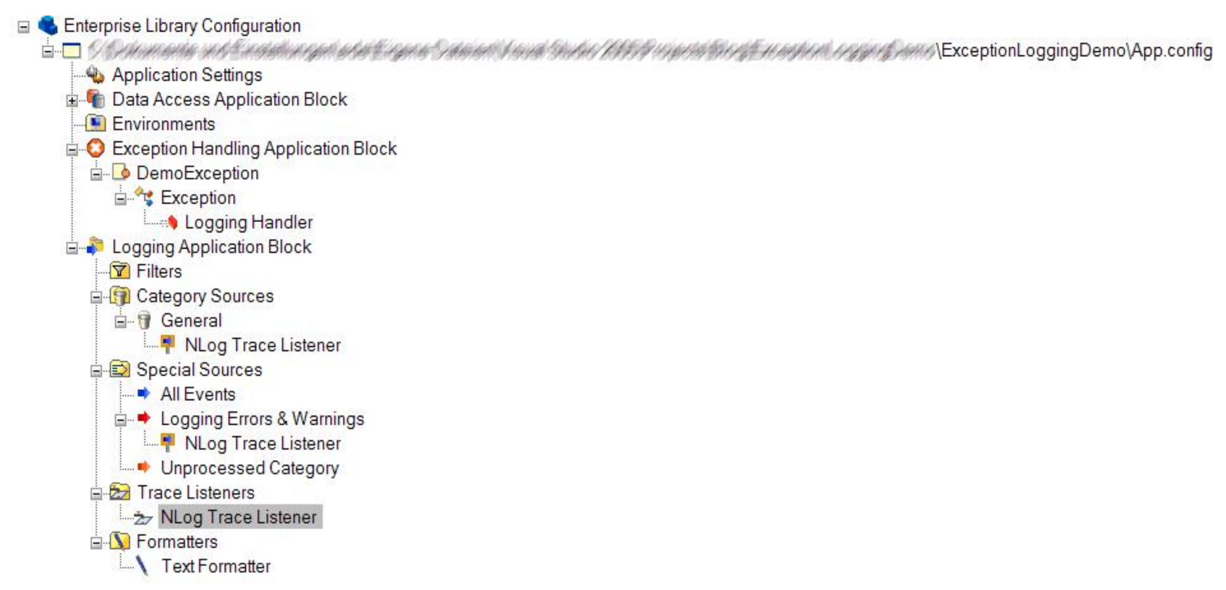


Abbildung 10: NLog Trace Listener

Als Verbesserung könnte hier ein eigener Formatter erstellt werden, welcher die Daten in einer besser lesbaren Form in unseren NLogTraceListener schreibt.

Hier nun der Download des Beispiel-Projektes:

Download NLog Trace Listener Demo

[1] Download Enterprise Library 3.1 (May 2007)

2.2.87.Rekursive Methoden mit Hilfe eines Stacks abbilden

Rekursionen werden recht gerne für unterschiedlichste Zwecke eingesetzt. Beispielsweise um ein bestimmtes Steuerelement auf einem Formular (oder darunterliegenden Containern) zu finden. Allerdings muss eine Rekursion nicht immer sein. Der Nachteil einer Rekursion besteht im zahlreichen Aufruf derselben Methode. Methodenaufrufe sind kostspielig und und sollten daher nicht übertrieben werden. In einigen Fällen der Rekursion läßt sich das vermeiden.

Im nachfolgenden Beispiel wird eine vorhandene Rekursion mit Hilfe eines Stacks ersetzt. Die Stack-Variante ist recht einfach umgesetzt und zudem auch noch performanter als die rekursive.

Um ein bestimmtes Steuerelement auf einem Formular zu finden, würde uns folgender rekursiver Ansatz zum Ziel bringen:

```
private Control FindRecursively(Control c, string strControlName)
{
    if (c.Name == strControlName)
        return c;
    foreach (Control cf in c.Controls)
    {
        Control cTemp = FindRecursively(cf, strControlName);
```

```
        if (cTemp != null)
            return cTemp;
    }
    return null;
}
```

Recht schnell erklärt: Das jeweils übergebene Control wird auf den Namen hin überprüft. Entspricht dieser nicht dem gewünschten, wird die Controls-Auflistung durchlaufen und dieselbe Methode wieder aufgerufen. Dies wird so lange gemacht bis entweder das gewünschte Steuerelement gefunden wurde, oder alle durchlaufen wurden. Das nachfolgende Beispiel zeigt, wie dies nicht-rekursiv, mit Hilfe eines Stacks gelöst werden kann:

```
private Control FindNotRecursively(Control c, string strControlName)
{
    if (c.Name == strControlName)
        return c;
    Stack stack = new Stack();
    stack.Push(c);
    while (stack.Count > 0)
    {
        Control cTemp = stack.Pop();
        if (cTemp.Name == strControlName)
            return cTemp;
        foreach (Control cf in cTemp.Controls)
            stack.Push(cf);
    }
    return null;
}
```

Auch diese Variante ist schnell erklärt: Das übergebene Steuerelement wird auf den Stack gelegt. Die darauffolgende Schleife durchläuft den Stack solange, solange sich Elemente darauf befinden. Per Pop wird immer ein Steuerelement vom Stack genommen und die Eigenschaft Name überprüft. Danach werden alle Steuerelemente aus der Controls-Auflistung auf den Stapel gelegt und ebenfalls durchlaufen. Dies geschieht ebenfalls so lange bis das gewünschte Steuerelement gefunden wurde, oder sich keine weiteren Elemente mehr auf dem Stapel befinden.

Ein kleiner Testfall wurde durch ein dynamisches Anlegen von Steuerlementen geschaffen:

```
Panel currentPanel = new Panel();
this.Controls.Add(currentPanel);
for (int i = 0; i < 40; i++)
{
    Panel p = new Panel();
    for (int i2 = 0; i2 < 80; i2++)
    {
        Label l = new Label();
        p.Controls.Add(l);
    }
    currentPanel.Controls.Add(p);
    currentPanel = p;
}
Button btn1 = new Button();
btn1.Name = "button1";
currentPanel.Controls.Add(btn1);
```

Danach wurden die unterschiedlichen Varianten mittels **Stopwatch** gemessen. Hier das Ergebnis:

```
// Zahlen in Millisekunden
// Erster Durchlauf
Recursively: 4
!Recursively: 1
// Folgende Durchläufe
Recursively: 1
!Recursively: 0
```

Die zahlen wirken auf den ersten Blick nicht sehr spektakulär. Zu beachten ist jedoch, dass die Beispielrekursion recht klein ist und wenig Aufwand verursacht. In der Realität sind viele Rekursionen meist aufwändiger, wodurch sich der Geschwindigkeitsunterschied verdeutlicht.

2.2.88.Generische Typ-Informationen auslesen

Wer viel mit Reflection arbeitet der bekommt es von Zeit zu Zeit natürlich auch mit generischen Datentypen zu tun. Vielfach wird die Frage gestellt, wie man hierbei an die Typ-Informationen der generischen Parameter gelangt. Die Lösung ist denkbar einfach:

```
Dictionary<int, string> dict = new Dictionary<int, string>();
Type dictType = dict.GetType();
Console.WriteLine("IsGenericType: " + dictType.IsGenericType);
foreach (Type t in dictType.GetGenericArguments())
    Console.WriteLine("Argument-Typ: " + t.FullName);
if (dictType.IsGenericParameter)
    foreach (Type t in dictType.GetGenericParameterConstraints())
        Console.WriteLine("Contraint: " + t.FullName);
Type typeDef = dictType.GetGenericTypeDefinition();
Console.WriteLine("TypeDefinition: " + typeDef.FullName);
```

Dieses kurze Beispiel zeigt, wie die Typen der generischen Parameter ausgelesen werden können.

So sieht das Ergebnis aus:

```
IsGenericType: True
Argument-Typ: System.Int32
Argument-Typ: System.String
TypeDefinition: System.Collections.Generic.Dictionary`2
```

Es lohnt sich in weiterer Folge, sich genauer mit den angebotenen Methoden und Eigenschaften der Klasse **Type** zu beschäftigen.

2.2.89.Häufigkeiten von Wörtern in einer Liste berechnen

Heute fand ich eine Frage vor, wie denn ein Array bestehend aus unterschiedlichen Namen durchlaufen werden kann, um die einzelnen Namen und die Anzahl der Vorkommen ausgeben zu lassen. Mein Vorschlag wäre an dieser Stelle eine eigens abgeleitete **List<string>**. Dies würde so aussehen:

```
public class NameCounterList : List<string>
{
    private Dictionary<string, int> _names =
        new Dictionary<string, int>();
    public new void Add(string item)
    {
        base.Add(item);
        if (_names.ContainsKey(item.ToLower()))
            _names[item.ToLower()]++;
        else
            _names.Add(item.ToLower(), 1);
    }
    public new void Clear()
    {
        base.Clear();
        _names.Clear();
    }
    public override string ToString()
    {
        StringBuilder sb = new StringBuilder();
        if (this._names.Count > 0)
        {
            Dictionary<string, int>.Enumerator en =
                this._names.GetEnumerator();
            while (en.MoveNext())
            {
                sb.Append(en.Current.Key);
                sb.Append(": ");
                sb.Append(en.Current.Value.ToString());
                sb.Append(System.Environment.NewLine);
            }
        }
        return sb.ToString();
    }
}
```

Die Aufrufe können dann wie folgt aussehen:

```
NameCounterList ncl = new NameCounterList();
ncl.Add("Peter");
ncl.Add("Harry");
ncl.Add("Sabine");
ncl.Add("Sabine");
ncl.Add("Jörg");
ncl.Add("Harry");
ncl.Add("Harry");
Console.WriteLine(ncl.ToString());
```

Wodurch sich folgender Output ergibt:

```
peter: 1
harry: 3
sabine: 2
jörg: 1
```

Anmerkungen
An dieser Stelle können natürlich noch Verbesserungen vorgenommen werden. Beispielsweise wäre es denkbar, die Werte zu sortieren. Auf der anderen Seite muss jedoch auch erwähnt werden, dass es für große Datenmengen durchaus bessere Möglichkeiten gibt, eine derartige Funktionalität abzubilden. Für kleinere Anwendungsfälle sollte diese Variante aber durchaus ausreichen.

2.2.90.C# 3.0 - Automatische Eigenschaften

Ein häufiges Konstrukt um unter C# 2.0 Eigenschaften zu erstellen, ist dieses hier:

```
private string myProperty;

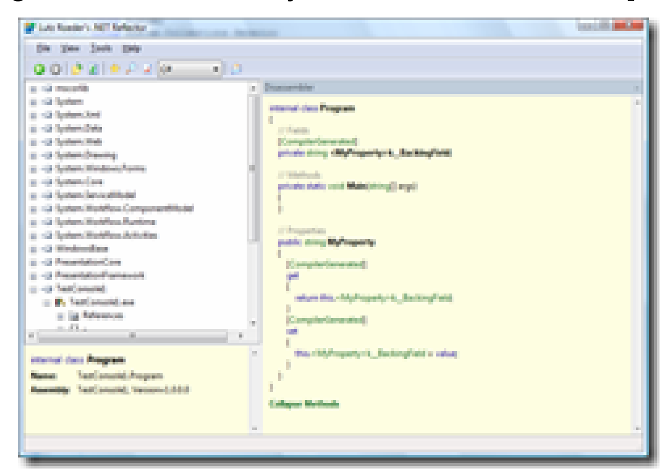
public string MyProperty
{
    get { return myProperty; }
    set { myProperty = value; }
}
```

In C# 3.0 kann eine solche Eigenschaft mit viel weniger geschriebenem Code erzeugt werden, genauer gesagt mit einer einzigen Zeile Code.

```
public string MyProperty { get; set; }
```

Das zugehörige Feld und der Getter bzw. Setter werden automatisch durch den Compiler erzeugt.

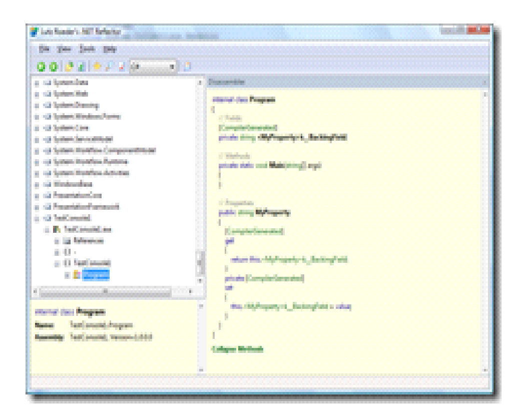
Dass dies tatsächlich so ist kann mit Hilfe des .NET Refletor überprüft werden. Der generierte Code ist jeweils mit dem Attribut *[CompilerGenerated]* gekennzeichnet.



Natürlich können automatische Eigenschaften immer nur mit Getter und Setter erzeugt werden. Damit der Setter nur innerhalb der Klasse benutzt werden kann, muss dieser einfach als private deklariert werden.

```
public string MyProperty { get; private set; }
```

Der generierte Code zeigt nun jeweils einen privaten Setter und öffentlichen Getter.



2.2.91.Ausgabe von Datum und Uhrzeit formatieren

Um das aktuelle Datum und die aktuelle Uhrzeit ausgeben zu lassen ist nichts weiter nötig als eine Zeile Code.

```
System.DateTime.Now.ToString(); //Beispiel:
   16.09.1981 13:44:51
```

Weiter ist es möglich mit den Methoden `ToShortDateString()`, `ToLongDateString()`, `ToShortTimeString()` und `ToLongTimeString()` jeweils das Datum oder die Uhrzeit in kurzer und langer Form auszugeben.

Möchte man allerdings das Datum z.B. im Format 16-09-1981 13:44:51 formatieren sind Einsteiger häufig versucht, dies per *Replace()* zu lösen.

Allerdings bietet C# eine Überladung der Methode *ToString()* an, sodass ein Format in Form eines String übergeben werden kann. Dieses Format wird mit Kürzeln angegeben. Das Kürzel "yyyy" steht z. B. für ein vierstelliges Jahr (Beispiel 2007), während das Kürzel "MM" den aktuellen Monat (Beispiel 06) ausgibt.

Das oben gezeigte Problem kann nun sehr einfach gelöst werden.

```
System.DateTime.Now.ToString("dd-MM-yyyy
   HH:mm:ss"); //Beispiel: 16-09-1981 15:46:34
```

Auf diese Weise sind viele verschiedene Ausgaben möglich, ohne den aufwendigen Umweg über String-Operationen gehen zu müssen. Eine komplette Übersicht der Format-Kürzel ist in der MSDN vorhanden.

2.3.Windows Forms

2.3.1. ComboBox als DropDownList kann kein Text gesetzt werden

Wer eine ComboBox verwendet und die Eigenschaft `DropDownStyle` auf `DropDownList` gesetzt hat, kann keinen Text mehr setzen. Dadurch entfällt auch die Möglichkeit, einen Default-Text zu setzen, wenn kein Item ausgewählt ist/wurde. Dem kann durch eine kurze und schnelle Ableitung leicht Abhilfe geschafft werden.

```
public partial class ComboBoxEx : ComboBox
{
    private Label _statusLabel = new Label();
    private string _statusText = null;

    public string StatusText
    {
        get { return this._statusText; }
        set { this._statusText = value; }
    }

    public ComboBoxEx()
    {
        InitializeComponent();

        Init();

        this.Controls.Add(this._statusLabel);

        this._statusLabel.Click += new EventHandler(_statusLabel_Click);
        this.SizeChanged += new EventHandler(ComboBoxEx_SizeChanged);
        this.SelectedIndexChanged += new
EventHandler(ComboBoxEx_SelectedIndexChanged);
    }

    void _statusLabel_Click(object sender, EventArgs e)
    {
        this.DroppedDown = true;
    }

    void ComboBoxEx_SelectedIndexChanged(object sender, EventArgs e)
    {
        if (this.SelectedItem == null)
            Init();
        else
            this._statusLabel.Visible = false;
    }

    void ComboBoxEx_SizeChanged(object sender, EventArgs e)
    {
        Init();
    }

    public void Init()
    {
        if (this.DropDownStyle == ComboBoxStyle.DropDownList)
        {
            this._statusLabel.Visible = true;
            this._statusLabel.Location = new Point(1, 1);
            this._statusLabel.Size = new Size(this.Width - 20, this.Height - 2);

            if (this._statusText != null)
            {
                this._statusLabel.Text = this._statusText;
                this._statusLabel.Font = this.Font;
            }
```

```
                else
                {
                    this._statusLabel.Text = "[Nothing selected]";
                }
                this._statusLabel.BringToFront();
            }
            else
            {
                this._statusLabel.Visible = false;
            }
        }
}
```

Es wird direkt von der `ComboBox` abgeleitet. Die neue Klasse erhält die Eigenschaft `StatusText` mit dessen Hilfe ein entsprechender Text gesetzt werden kann, der angezeigt wird, wenn kein Item ausgewählt ist.

2.3.2.Meine GUI friert während der Ausführung ein, was tun?

Ebenfalls eine oft gesehene Frage. Eine Aufgabe wird gerade ausgeführt und dabei friert die Oberfläche der Anwendung ein. Weder ein Fortschrittsbalken wird aktualisiert, noch ein Label, das den Fortschritt anzeigt. Ja selbst beim Verschieben der Form wird diese weiß und zeigt keine Informationen mehr an.

Der Hintergrund liegt einfach daran, dass durch eine ausgeführte Arbeit von dieser Form keine System-Message mehr angenommen wird bzw. diese nicht verarbeitet werden kann. Um dies zu vermeiden (und damit der Benutzer nicht irrtümlich annimmt, die Anwendung sei abgestürzt), bieten sich mehrere Lösungen an:

DoEvents
Durch ein `Application.DoEvents()` erhält die Anwendung die Möglichkeit, andere Events abzuhandeln und die Form neu zu zeichnen.

Threading
Aufwändige Aufgaben sollten in einem eigenen Thread ausgeführt werden. Dies kann über die Klassen des Namespaces `System.Threading` geschehen oder über einen `BackgroundWorker`. Hier ist jedoch zu beachten, dass von Threads nicht direkt auf die GUI-Elemente zugegriffen werden kann. Hierzu muss mit `Invoke` gearbeitet werden.

Wichtig: Locking
In manchen Fällen ist es wichtig, bestimmte Code-Teile mit einem `lock` (oder anderen entsprechenden Blockierungs-Maßnahmen) zu versehen. Dadurch kann der gleiche Code nur von einem einzigen Thread aufgerufen und ausgeführt werden. Dies kann mit einem `DoEvents` nicht ausreichend gelöst werden. In solchen Fällen sollte auf jeden Fall Threading verwendet werden.

2.3.3.Menüs dynamisch mit Hilfe einer XML Datei erstellen

In manchen Fällen macht es Sinn, bestimmte Teile einer Anwendung dynamisch erstellen zu lassen. Ob es sich dabei um Menüs handelt, um Eingabefelder, oder

auch anderen Dingen. Immer wieder stößt man auf entsprechende Fragen, oft mit keinem wirklich guten Beispiel, vor allem für .NET Newcomer.

Hier nun ein Beispiel für das dynamische Erstellen eines Menüs aus einem XML File heraus.

Wie funktioniert das?
Die XML-Datei beschreibt die Hauptmenü-Einträge als auch die Untermenüs. Informationen wie Name, Text, welche Methode welcher Klasse bei einem Klick aufgerufen werden soll, sowie ob der Menüeintrag beim Startup der Anwendung verfügbar ist, werden darüber gesteuert.

Für jeden Hauptmenü-Eintrag existiert eine eigene Klasse, welche die entsprechenden Methoden zur Verfügung stellt. Für diese werden über den `DynamicMenuHandler` automatisiert Delegates erstellt, die das `Click`-Ereignis an diese Methoden weiterleiten und den dahinterliegenden Code ausführen.

Die gleiche Vorgehensweise kann auch für andere Zwecke verwendet (missbraucht) werden.

Download Dynamic Menu Creator Beispiel

Sollten zu diesem Beispiel Fragen auftauchen, dann können diese natürlich über die Kommentar-Funktion gestellt werden. Der Download liegt als Visual Studio 2005 Solution vor.

2.3.4.UserControls eines Namespaces finden

Auf die Frage hin, wie man denn alle UserControls eines bestimmten Namespaces herausfinden kann, um diese dann dynamisch in ein Container-Control zu verfrachten, schrieb ich eine kleine Testanwendung die zeigt, wie man alle Klassen aus einem bestimmten Namespace bekommt. Zusätzlich wird die Information ausgegeben, ob es sich dabei um ein UserControl handelt oder nicht.

Download Beispiel-Projekt (VS 2005 Solution, 40KB)

Bei dieser Lösung ist jedoch anzumerken, dass sich ein Namespace auch über mehrere Assemblies hinweg erstrecken kann. Dies wurde nicht berücksichtigt. Das Beispiel sollte auch eher einen Denkanstoss geben, als eine fix fertige Solution liefern.

2.3.5..NET 2.0: ComboBox und AutoComplete

Unter .NET 2.0 gibt es eine sehr einfache Möglichkeit, der ComboBox eine `AutoComplete`-Funktion bzw. eine Vorschlags-Funktion zu verpassen.

Dazu einfach folgende Schritte ausführen:

1. `AutoCompleteCustomSource` setzen (zusätzlich zur normalen `DataSource`)
2. `AutoCompleteMode` auf `Suggest` stellen
3. `AutoCompleteSource` auf `ListItems` stellen

Fertig ist die Hexerei und der User freut sich über die verbesserte Usability. Statt `Suggest` (Punkt 2) gibt es auch noch weitere Möglichkeiten um eventuell eigene Einträge anzuhängen etc. Einfach ein wenig ausprobieren.

2.3.6. C# Beginner: UserControl DoubleTrackBar Beispiel

Deisem Eintrag liegt ein Beispiel bei, welches zeigt, wie ein einfaches `DoubleTrackBar`-Control erstellt werden kann.

Mit Hilfe dieses Controls können mit zwei Schiebereglern ein Minimum-Wert und ein Maximum-Wert eingestellt werden. Eigentlich ein recht simples Problem, jedoch nicht für C# Programmierer, die in Themen wie GDI+, UserControls wenig Erfahrung haben.

Der folgende Screenshot zeigt das Aussehen des UserControls in einer kleinen Testanwendung:

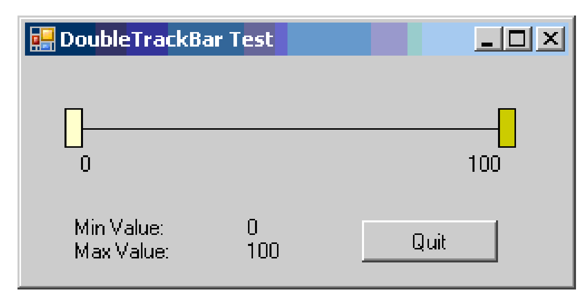


Abbildung 11: Double Trackbar

Zusätzliche Funktionalitäten sollten sich recht einfach einfügen lassen.

Anforderungen: .NET 2.0 für die Solution. Sourcecode auch unter .NET 1.1 nutzbar.

Download

2.3.7. C# Beginner: Beispiel für den Aufbau eines Strategie-Spieles

In diversen Foren wird oft nachgefragt, wie denn man denn ein Strategiespiel angehen könnte. Im Vordergrund stehen dabei keine Probleme á la Pathfinding, sondern bereits das Aufbauen des Spielfeldes bereitet oft Probleme.

Daher habe ich aus Lust und Laune eine kleine Demo erstellt, aus der man diverse Ansätze herauslesen kann.

Das Aussehen der Demo hatte hierbei keinen Vorrang und kommt daher mit folgender Oberfläche daher:

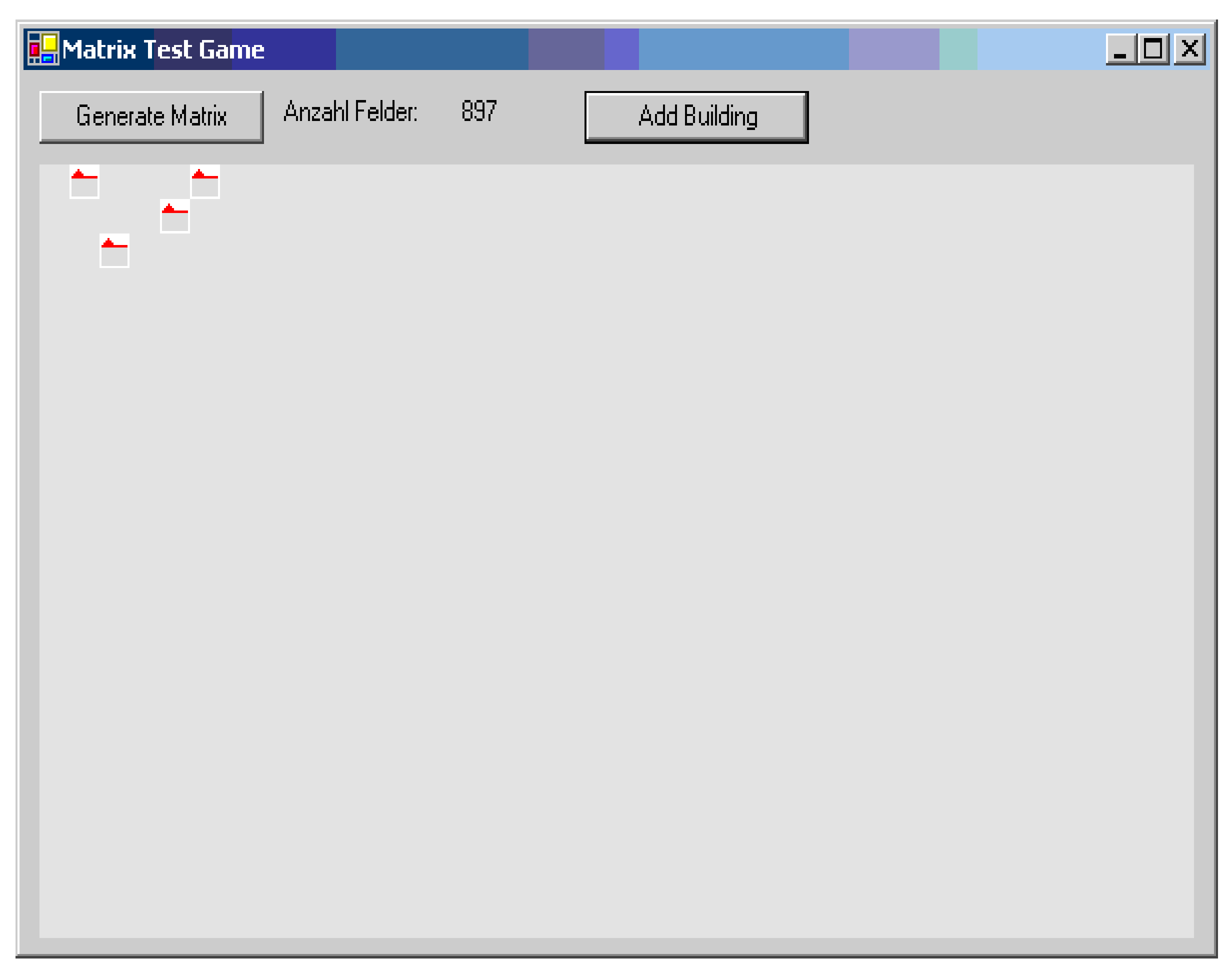


Abbildung 12: Matrix Test Game

Nun zur Erklärung einige Punkte:

Darstellungsfläche stellt in diesem Beispiel ein ganz normales Panel dar, welches lediglich zum besseren Erkennen eingefärbt wurde. Wird nun der Button **Generate Matrix** betätigt, wird quasi eine Matrix auf die Spielfläche gelegt. Diese wird automatisch berechnet und kann durch den Parameter `Length` gesteuert werden. Dieser gibt die Seitenlänge des Quadrates an.

Der Button **Add Building** dient dazu, ein Building anzulegen. Es ist so vorzugehen, dass der Button zu klicken ist, danach ist auf eine beliebige Stelle des Panels zu klicken. Ist der Platz noch nicht belegt, wird das Gebäude auf diesem Feld "gebaut", andernfalls erscheint die entsprechende Meldung.

Dies ist im Endeffekt auch schon die gesamte Funktionalität dieser Demo. Trotz der Kürze der Entwicklungszeit habe ich dennoch versucht, dies möglichst einfach und erweiterbar zu gestalten, so dass aufgrund dieser "Vorlage" recht schnell unterschiedliche Bauten etc. eingefügt werden können.

Vorstellbar wären sicherlich noch Dinge wie Wiesen, Personen und auch Gebäude,

die nicht auf ein Feld beschränkt sind. Ebenfalls müsste noch ein Pathfinding-Algorithmus implementiert werden und danach wäre eine erste spielbare Demo fast fertig.

Sollte ich Zeit finden werden noch die einen oder anderen Erweiterungen in diese Demo fließen, aber grundlegend sollte eine Basis für simple Strategie-Spiele vorhanden sein.

Hier noch das Demoprojekt inklusive Sourcecode: MatrixTestGame

2.3.8.Transparente Steuerelemente mit C#

Eigentlich ein alter Hut, aber ad hoc ist es mir heute auch nicht eingefallen wie eigene Steuerelemente transparent erstellt werden können. Der erste Versuch mit

```
this.Background = Color.Transparent;
```

scheiterte kläglich. Hier ein Lösungsweg:

```
protected override CreateParams CreateParams
{
    get
    {
        CreateParams cp = base.CreateParams;
        cp.ExStyle := 0x20;
        return cp;
    }
}

protected override void OnPaintBackground
    (PaintEventArgs pevent)
{
    // do nothing in this case
}

protected override void OnMove(EventArgs e)
{
    RecreateHandle();
}
```

2.3.9.Controls auf einem Formular bewegen

In Foren als auch meiner Inbox taucht immer wieder die Frage auf, wie man denn bewegbare Controls erstellen kann, um beispielsweise ein Diagramm zu zeichnen etc. Deshalb möchte ich hier ein ganz kleines Beispiel zeigen, wie in zwei Minuten eine bewegliche Basisklasse für bewegliche Controls erstellt werden kann. Und hier kommt schon der Sourcecode:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
```

```
using System.Data;
using System.Text;
using System.Windows.Forms;
namespace MoveableControlDemo
{
    public partial class MoveableBaseControl
        : UserControl
    {
        private bool _isMoving = false;
        private int _deltaX = 0;
        private int _deltaY = 0;
        public MoveableBaseControl()
        {
            InitializeComponent();
        }
        private void MoveableBaseControl_MouseDown(
            object sender,
            MouseEventArgs e)
        {
            _isMoving = true;
            _deltaX = e.X;
            _deltaY = e.Y;
        }
        private void MoveableBaseControl_MouseUp(
            object sender,
            MouseEventArgs e)
        {
            _isMoving = false;
        }
        private void MoveableBaseControl_MouseMove(
            object sender,
            MouseEventArgs e)
        {
            if (_isMoving && e.Button == MouseButtons.Left)
            {
                this.Location = new Point(
                    this.Location.X + (e.X - _deltaX),
                    this.Location.Y + (e.Y - _deltaY)
                    );
            }
        }
    }
}
```

Im Grunde passiert nicht viel. Es wird lediglich festgehalten wann sich das Control bewegen soll und wann nicht (`MouseDown` und `MouseUp`). Bei einem `MouseMove` wird anschließend die Position des Controls berechnet. Zu beachten ist hier nur, dass die `MouseEventArgs` die Position des Cursors innerhalb des Controls angibt und nicht auf Formular- bzw. Screen-Basis. Daher muss das Delta zum Rand des Controls beim `MouseDown` festgehalten werden, um dies später in die Berechnung einfließen zu lassen.

Dieses Control kann sozusagen als ein Basis-Control verwendet werden. Zum Test einfach in ein Projekt einbauen, auf ein Formular ziehen, Anwendung starten und mit der Maus über das Formular bewegen.

2.3.10.AutoScroll für RichTextBox

In der letzten Zeit wurde ich des öfteren gefragt, wie man denn eine `RichTextBox` dazu bringt, automatisch zu scrollen, sobald neuer Text hinzugefügt wird (wenn diese beispielsweise als Ausgabe für Log-Informationen verwendet wird). Hier wie ich es mache:

```
this.LogRtb.Text += "My additional log text ...";
this.LogRtb.SelectionStart = this.LogRtb.Text.Length;
this.LogRtb.ScrollToCaret();
```

2.3.11. Mehrzeiliges Editieren von String-Eigenschaft im Eigenschaften Fenster

Unter Visual Studio 2005 (.NET 2.0) ist es möglich, Properties des Typs String mehrzeilig zu editieren:



Abbildung 13: Mehrzeiliges Editieren

Erstellt man ein benutzerdefiniertes Steuerelement, sollen eigene String-Eigenschaften idealerweise ebenfalls mit dieser Hilfe ausgestattet werden. Hierzu ist ein entsprechendes Attribut zu setzen:

```
[Localizable(true),
 Editor("System.ComponentModel.Design.MultilineStringEditor," +
     "System.Design, Version=2.0.0.0, Culture=neutral, " +
     "PublicKeyToken=b03f5f7f11d50a3a",
 typeof(UITypeEditor))]
public string Description
{
    get { return this.DescriptionLabel.Text; }
    set { this.DescriptionLabel.Text = value; }
}
```

Ab sofort bietet das benutzerdefinierte Steuerelement ebenfalls dieses Feature an und mehrzeiliger Text kann bequem eingegeben werden.

2.3.12. UserControls im Skype-Stil selbst erstellt

Beiträge zur Erstellung von UserControls finden sich im Internet wahrlich viele. Dennoch wird immer wieder danach gefragt, wodurch ich mich schließlich hinreissen ließ, eine kurze Demo zu erstellen. Gezeigt wird, wie ein ProgressBar im Skype-Stil erstellt wird.

Grundlegend ist eine neue Klasse zu erstellen, welche von **UserControl** ableitet. Da

das Control via GDI+ gezeichnet wird, muss an dieser Stelle das **OnPaint**-Event überschrieben werden, was wir entsprechend im Konstruktor mitteilen müssen:

```
this.SetStyle(ControlStyles.UserPaint, true);
this.SetStyle(ControlStyles.OptimizedDoubleBuffer, true);
this.SetStyle(ControlStyles.AllPaintingInWmPaint, true);
```

Wird `ControlStyles.UserPaint` auf `true` gesetzt, müssen alle notwendigen Aktualisierungen der Oberfläche selbst vorgenommen werden. Die beiden weiteren Styles sind lediglich Hilfen, die ein Flackern beim Neuzeichnen verhindern bzw. unterdrücken sollen.

Ist dieser Schritt getan, muss die Logik implementiert werden. Dazu gehören die entsprechenden Eigenschaften zur Bestimmung des Minimums, Maximums und des aktuellen Values (wie wir es von einer Standard-ProgressBar gewohnt sind). Wurde auch dies erledigt, geht es daran, das Control zu zeichnen.

Hierzu werden ich allerdings den gesamten Sourcecode des UserControls auflisten:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Text;
using System.Windows.Forms;
using System.Drawing.Drawing2D;
namespace SkypeLookControls
{
    public partial class SkypeProgressBar : UserControl
    {
        private int _min = 0;
        private int _max = 100;
        private int _value = 0;
        private Point[] _borderPoints = null;
        private Color borderColor = Color.FromArgb(174, 179, 179);
        private Color brightDisabledColor = Color.FromArgb(222, 222, 222);
        private Color darkDisabledColor1 = Color.FromArgb(209, 209, 209);
        private Color darkDisabledColor2 = Color.FromArgb(216, 216, 216);
        private Color brightActiveColor = Color.FromArgb(0, 183, 254);
        private Color darkActiveColor1 = Color.FromArgb(0, 167, 233);
        private Color darkActiveColor2 = Color.FromArgb(0, 183, 254);
        /// <summary>
        /// Gets or sets the minimum.
        /// </summary>
        /// <value>The minimum.</value>
        public int Minimum
        {
            get { return this._min; }
            set { this._min = value; }
        }
        /// <summary>
        /// Gets or sets the maximum.
        /// </summary>
        /// <value>The maximum.</value>
        public int Maximum
        {
            get { return this._max; }
            set { this._max = value; }
        }
        /// <summary>
        /// Gets or sets the value.
        /// </summary>
```

```
/// <value>The value.</value>
public int Value
{
    get { return this._value; }
    set { this._value = value; this.Invalidate(); this.Update(); }
}
/// <summary>
/// Initializes a new instance of the
/// <see cref="SkypeProgressBar"/> class.
/// </summary>
public SkypeProgressBar()
{
    InitializeComponent();
    this.SetStyle(ControlStyles.UserPaint, true);
    this.SetStyle(ControlStyles.OptimizedDoubleBuffer, true);
    this.SetStyle(ControlStyles.AllPaintingInWmPaint, true);
    Init();
    this.Resize += new EventHandler(SkypeProgressBar_Resize);
}
void SkypeProgressBar_Resize(object sender, EventArgs e)
{
    Init();
    this.Invalidate();
    this.Update();
}
private void Init()
{
    this._borderPoints = new Point[9];
    Point p1 = new Point(3, 1);
    Point p2 = new Point(this.Width - 3, 1);
    Point p3 = new Point(this.Width - 1, 3);
    Point p4 = new Point(this.Width - 1, this.Height - 3);
    Point p5 = new Point(this.Width - 3, this.Height - 1);
    Point p6 = new Point(3, this.Height - 1);
    Point p7 = new Point(1, this.Height - 3);
    Point p8 = new Point(1, 3);
    Point p9 = new Point(3, 1);
    this._borderPoints[0] = p1;
    this._borderPoints[1] = p2;
    this._borderPoints[2] = p3;
    this._borderPoints[3] = p4;
    this._borderPoints[4] = p5;
    this._borderPoints[5] = p6;
    this._borderPoints[6] = p7;
    this._borderPoints[7] = p8;
    this._borderPoints[8] = p9;
}
/// <summary>
/// Raises the
/// <see cref="E:System.Windows.Forms.Control.Paint"></see>
/// event.
/// </summary>
/// <param name="e">A
/// <see cref="T:System.Windows.Forms.PaintEventArgs"></see>
/// that contains the event data.</param>
protected override void OnPaint(PaintEventArgs e)
{
    base.OnPaint(e);
    Point[] points = new Point[4];
    Point p1 = new Point(0,0);
    Point p2 = new Point(this.Width, 0);
    Point p3 = new Point(this.Width, this.Height);
    Point p4 = new Point(0, this.Height);
    points[0] = p1;
    points[1] = p2;
    points[2] = p3;
    points[3] = p4;
    e.Graphics.DrawPolygon(
        new Pen(new SolidBrush(this.BackColor)), points);
    PaintBorder(e);
```

```
        PaintBar(e);
    }
    /// <summary>
    /// Paints the border.
    /// </summary>
    /// <param name="e">The
    /// <see cref="System.Windows.Forms.PaintEventArgs"/>
    /// instance containing the event data.</param>
    private void PaintBorder(PaintEventArgs e)
    {
        Pen p = new Pen(new SolidBrush(this.borderColor));
        e.Graphics.DrawLines(p, this._borderPoints);
    }
    /// <summary>
    /// Paints the disabled part of the progress bar.
    /// </summary>
    /// <param name="e">The
    /// <see cref="System.Windows.Forms.PaintEventArgs"/>
    /// instance containing the event data.</param>
    private void PaintDisabledPart(PaintEventArgs e)
    {
        int y = (this.Height-6) / 2;
        LinearGradientBrush lgb =
            new LinearGradientBrush(
                new Rectangle(3,y, this.Width-5, this.Height-y-3),
                this.darkDisabledColor1,
                this.darkDisabledColor2,
                90f);
        e.Graphics.FillRectangle(lgb,
                new Rectangle(3, y+1, this.Width - 5, this.Height-y-3));
        e.Graphics.FillRectangle(
                new SolidBrush(this.brightDisabledColor),
                new Rectangle(3, 3, this.Width - 5, y));
    }
    /// <summary>
    /// Paints the active part of the progress bar.
    /// </summary>
    /// <param name="e">The
    /// <see cref="System.Windows.Forms.PaintEventArgs"/>
    /// instance containing the event data.</param>
    private void PaintActivePart(PaintEventArgs e)
    {
        int y = (this.Height - 6) / 2;
        double width = (this.Width - 6d) /
             (double) this.Maximum * (double)this.Value;
        if (Value == Maximum)
            width = this.Width - 6d;
        if (width > 0)
        {
            LinearGradientBrush lgb =
                new LinearGradientBrush(
                    new Rectangle(3, y, (int)width, this.Height - y - 3),
                    this.darkActiveColor1,
                    this.darkActiveColor2,
                    90f);
            e.Graphics.FillRectangle(lgb,
                new Rectangle(3, y + 1, (int)width, this.Height - y - 3));
            e.Graphics.FillRectangle(
                new SolidBrush(this.brightActiveColor),
                new Rectangle(3, 3, (int)width, y));
        }
    }
    /// <summary>
    /// Paints the progress bar.
    /// </summary>
    /// <param name="e">The
    /// <see cref="System.Windows.Forms.PaintEventArgs"/>
    /// instance containing the event data.</param>
    private void PaintBar(PaintEventArgs e)
    {
```

```
                PaintDisabledPart(e);
                PaintActivePart(e);
            }
        }
    }
```

Wichtig ist, dass beim Setzen der Eigenschaft **`Value`** durch ein **`Invalidate`** das Neuzeichnen des Controls veranlasst wird. Durch den Aufruf der **`Update`**-Methode, wird die Aktualisierung sofort vorgenommen. Durch den Auftrag, das Control neu zu zeichnen, wird das **`OnPaint`**-Event ausgelöst, welches nun das Control tatsächlich via GDI+ zeichnet.

Das war es mit der "Hexerei".

Das Endergebnis sieht schließlich folgendermaßen aus:

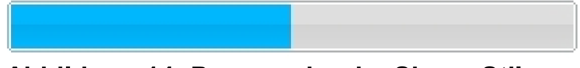


Abbildung 14: Progressbar im Skype-Stil

2.4.ASP.NET

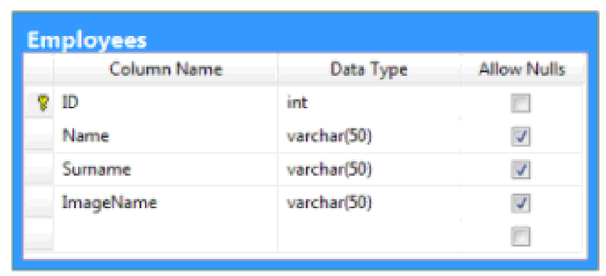
2.4.1.ViewState und TextBox - Control

Das Attribut `EnableViewState` gibt an, ob der Zustand des zugehörigen Controls im ViewState gespeichert werden soll. Dies ist natürlich auch bei dem TextBox Control der Fall. Allerdings ist nach einem PostBack der Zustand noch immer vorhanden, obwohl das Attribut auf false gesetzt wurde. Der Grund hierfür ist das PostBack selbst. Wird ein Formular abgeschickt, werden die Daten per POST oder GET an die Zielseite geschickt. Da die Zielseite gleich der Quellseite ist, stehen somit die Daten auch über PostBacks hinweg zur Verfügung.
Möchte man den Zustand des Controls explizit zurücksetzen, muss dies manuell erledigt werden.

2.4.2.Bilder im GridView anzeigen

Bilder im GridView anzuzeigen erscheint den meisten erfahrenen ASP.NET Usern als ziemlich einfach. Dennoch wird diese Frage häufig in Foren gestellt. Zusammenfassend kann man sagen dass es wirklich mehr als einfach ist, wenn man nur die richtigen Schritte kennt.

Dieses Beispiel geht davon aus, dass die Bilder im Dateisystem vorliegen und der Name in einem Feld der Datenbank gespeichert ist.



Employees

	Column Name	Data Type	Allow Nulls
🔑	ID	int	☐
	Name	varchar(50)	☑
	Surname	varchar(50)	☑
	ImageName	varchar(50)	☑
			☐

Abbildung 15: Bilder im GriedView 1

Nachdem eine Verbindung zur Datenbank per `SqlDataSource` Control hergestellt wurde, können die angezeigten Felder des GridView über das Kontextmenü und den Punkt *Edit Columns* konfiguriert werden. In diesem Dialog muss ein `ImageField` Control hinzugefügt werden.

Über die Eigenschaften `DataImageUrlField` und `DataImageUrlFormatString` kann nun eingestellt werden, welches Datenbankfeld verwendet werden soll und wo die Bilder im Web vorliegen. Der Platzhalter `{0}` im Wert der Eigenschaft `DataImageUrlFormatString` wird vom GridView Control automatisch mit dem Inhalt des Feldes `DataImageUrlField` ersetzt, so dass ein gültiger Verweis auf ein Bild ausgegeben wird.

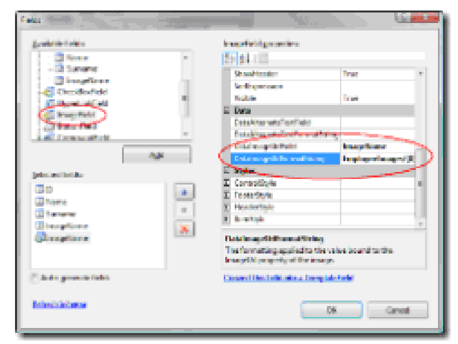


Abbildung 16: Bilder im GridView 2

Das Ergebnis ist ein zusätzliches Feld in Form einer Grafik. Den Download des Beispielprojekts findet man hier.

2.4.3.Mehrere Spalten im DataTextField der DropDownList

Leider ist es nicht möglich in der Eigenschaft `DataTextField` des DropDownList Controls mehr als eine Spalte anzugeben. Allerdings kann man mit Hilfe eines kleinen SQL-Tricks diese Gegebenheit umgehen.

```
SELECT *, (Name + ' ' + Surname) AS FullName FROM [Persons]
```

Diese Syntax fasst zwei vorhandene Spalten zusammen und gibt diese unter dem Namen `FullName` im Ergebnis aus. Eingebaut in das `SqlDataSource` Control, kann diese neue Spalte auch schon verwendet werden.

```
<asp:SqlDataSource ID="SqlDataSource1" runat="server"
    ConnectionString="<%$ ConnectionStrings:ConnectionString %>"
```

```
    SelectCommand="SELECT *, (Name + ' ' + Surname) AS FullName FROM [Persons]">
</asp:SqlDataSource>

<asp:DropDownList ID="DropDownList1" runat="server" DataSourceID="SqlDataSource1"
    DataTextField="FullName" DataValueField="ID">
</asp:DropDownList>
```

Das Beispielprojekt kann hier heruntergeladen werden.

2.4.4.Länge eines Strings per Validation Control überprüfen

Leider ist es mit dem ASP.NET `RangeValidator` nicht ohne weiteres möglich die Länge eines Strings zu überprüfen. Mit Hilfe eines kleinen regulären Ausdrucks und dem RegularExpressionValidator kann man diesen Umstand allerdings umgehen.

```
^\w{1,20}$
```

Dieser Ausdruck lässt nur Eingaben mit der maximalen Länge von 20 Zeichen zund Eingebaut in den `RegularExpressionValidator`, ist das Problem auch schon gelöst.

```
<asp:RegularExpressionValidator ID="regexCheck"
    runat="server"
    ErrorMessage="Error!"
    ValidationExpression="^\w{1,20}$" />
```

Möchte man die Eingabe von vornherein begrenzen, kann man das Attribut `MaxLength` des TextBox Controls benutzen.

```
<asp:TextBox runat="server" MaxLength="20" />
```

2.4.5.Per Shift-Taste mehrere CheckBox Controls markieren

Die Aufgabe ist es, ein wenig Windows Look & Feel in eine WebForm zu portieren. Genauer gesagt geht es darum mit Hilfe der Maus und der Shift Taste mehrere CheckBox Controls zu markieren. In diesem Beispiel ist eine Liste mit mehreren Produkten gegeben, die jeweils anhand einer CheckBox zum Löschen markiert werden können. Klickt man nun die erste CheckBox der Liste an, hält die Shift-Taste gedrückt und klick anschließend die letzte CheckBox werden automatisch alle zwischenliegenden Controls markiert.

ProductID	ProductName	ProductPrice	
1	Item 1	12,99	☐
2	Item 2	15,99	☐
3	Item 3	16,99	☐
4	Item 4	17,99	☐

Abbildung 17: Mehrere CheckBox Controls markieren

Diese Funktionalität soll per JavaScript umgesetzt werden, um ein PostBack und somit einen erneuten Aufbau der Liste zu vermeiden.

Zunächst ist es nötig, eine Liste mit allen vorhandenen CheckBox-Controls IDs aufzubauen. Hierfür bietet sich ein Array an, welches später mit Hilfe einer for-Schleife durchlaufen werden kann.

```
protected void GridView1_DataBound(object sender, EventArgs e)
{
    string js = "chkCollection = new Array(";
    int index = 0;

    foreach (GridViewRow row in GridView1.Rows)
    {
        CheckBox chkBox = (CheckBox)row.Cells[3].FindControl("chkBox");
        js += "'" + chkBox.ClientID + "',";

        chkBox.Attributes.Add("onclick", "ShiftClick(" + index + ");");
        index++;
    }

    js = js.Substring(0, js.Length - 1) + ");";

    ClientScript.RegisterClientScriptBlock(this.GetType(), "ClickScript", js,
true);
}
```

Bei dieser Gelegenheit wird jedem CheckBox-Control die Methode `ShiftClick` hinzugefügt, welche beim Event `onclick` angerufen werden soll und als Parameter, die Position innerhalb der Arrays entgegen nimmt.

Um die gedrückte Shift-Taste zu erkennen ist es nötig, dem WebForm entsprechende Methoden für die Events `onkeydown` und `onkeyup` hinzuzufügen. Wird eine Taste gedrückt, liefern diese Events einen KeyCode zurück der jeweils einer Taste zugeordnet ist. Die Shift Taste gibt den Code 16 zurück. Aufgabe dieser Methoden ist es nun, den `KeyCode` in der globalen Variable `globalKeyCode` zu speichern oder bei loslassen der Taste wieder zu löschen.

```
function Document_KeyDown(event)
{
    document.onkeydown = function SetGlobalKeyCode(event)
    {
        globalKeyCode = GetKeyCode(event);
    }
}

function Document_KeyUp(event)
{
    document.onkeyup = function ClearKeyCode(event)
    {
        globalKeyCode = 0;
    }
}
function GetKeyCode(event)
{
   event = event || window.event;
   return event.keyCode;
}
```

Wie oben angedeutet, wird die Methode `ShiftClick` aufgerufen sobald ein CheckBox Control geklickt wird. Diese Methode stellt zunächst mit Hilfe weiterer globaler Variablen (`firstClick` und `lastClick`) fest, ob bereits eine CheckBox vorher markiert wurde und die Shift-Taste gedrückt ist. Ist dies der Fall, wird

anschließend der Start und das Ende der Schleife ermittelt um die zwischenliegenden Controls zu markieren oder die Markierung zu entfernen.

```
function ShiftClick(clickedPos)
{
    lastClick = clickedPos;

    if(globalKeyCode == 16)
    {
        if(firstClick > -1 && lastClick > -1)
        {
            var i;
            var start = firstClick < lastClick ? firstClick : lastClick;
            var end = firstClick > lastClick ? firstClick : lastClick;

            for(i = start; i <= end; i++)
            {
                document.getElementById(chkCollection[i]).checked =
document.getElementById(chkCollection[lastClick]).checked;
            }
        }
    }

    firstClick = clickedPos;
}
```

Mit Hilfe dieser paar Zeilen wird dem User ein wenig mehr Komfort bei der Markierung der CheckBox-Controls geboten. Grade bei langen Listen erleichtert dies, die Arbeit erheblich. Zusätzlich zu dem gezeigten Feature sollte noch eine weitere CheckBox, die grundsätzlich alle Controls markiert, implementiert werden.

Den Download des Beispielprojekts findet man hier.

2.4.6. „Wirklich löschen?“ im DetailsView

In einem anderen Beitrag hatte ich bereits gezeigt, wie leicht es ist eine "Wirklich löschen?" Abfrage im GridView zu implementieren. Nun bekam ich mehrere Anfragen per E-Mail, ob es auch möglich ist diese Abfrage im DetailsView-Control einzufügen.

Natürlich ist dies möglich und ebenfalls sehr leicht zu realisieren. Schlüssel ist hier auch der Befehl *Convert this field into a TemplateField*, welcher im Dialog *Field* zur Verfügung steht.

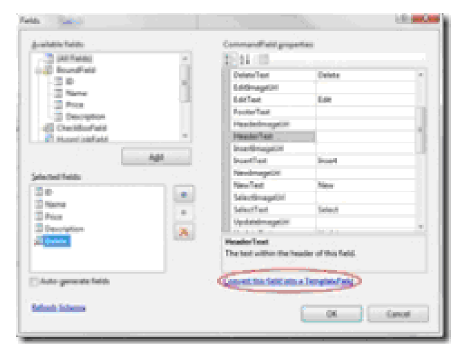


Abbildung 18: "Wirklich löschen" im DetailsView 1

Nach der Konvertierung steht ein vollwertiges LinkButton Control zur Verfügung. Das

Attribut OnClientClick des Controls kann nun dazu benutzt werden um eine JavaScript Anweisung auszuführen, sobald der Button geklickt wird.

```
<asp:TemplateField ShowHeader="False">
    <ItemTemplate>
        <asp:LinkButton ID="LinkButton1" runat="server" CausesValidation="False"
CommandName="Delete"
            Text="Delete" OnClientClick="return confirm('Wirklich
löschen');"></asp:LinkButton>
    </ItemTemplate>
</asp:TemplateField>
```

Wird nun der Link-Button angeklickt, öffnet sich ein Fenster und fordert den User nochmals zur Bestätigung der Anweisung auf.



Abbildung 19: "Wirklich löschen" im DetailsView 2

Der Rückgabewert des Dialogs wird per *return* weitergereicht und beendet ggf. die weitere Verarbeitung.

Das Beispielprojekt kann hier runtergeladen werden.

2.4.7.Login Control ohne ReturnUrl

Verwendet man Forms Authentication in Verbindung mit den Login-Controls von ASP.NET gibt es ein kleines Feature, dessen Funktion unter bestimmten Umständen unterbunden werden soll. Fordert der User eine Seite an, ohne sich authentifiziert zu haben, wird nach erfolgreichem Login auf diese Seite weitergeleitet. Die angeforderte Seite wird dabei per GET mit dem Key ReturnUrl übergeben.

Beispiel:
http://localhost/Login/default.aspx?ReturnUrl=%2fLogin%2fMember%2fMember.aspx

Allerdings gibt es viele Anwendungszwecke, bei denen dieses Feature eher hinderlich oder nicht erwünscht ist. Um diese Funktion zu deaktivieren, sind ein paar kleine Schritte nötig. Nach dem erfolgreichen Login, wird von dem Login-Control zunächst ein Event `LoggedIn` aufgerufen.

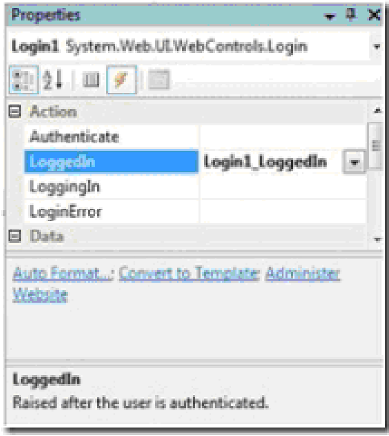


Abbildung 20: Login Control ohne ReturnUrl

Dieser Event muss nun mit einer entsprechenden Methode belegt werden deren Aufgabe es ist, die Weiterleitung per `Response.Redirect()` zu steuern.

```
protected void Login1_LoggedIn(object sender, EventArgs e)
{
    Response.Redirect("Default2.aspx");
}
```

Anschließend wird, unabhängig der `ReturnUrl`, immer auf die gewünschte Seite weitergeleitet.

Den Download des Beispielprojekts findet man hier.

2.4.8.Literal content is not allowed within a 'skin file'.

Zumindest auf den ersten Blick ist diese Fehlermeldung ein wenig verwirrend, deshalb eine kurze Erklärung dazu:

Die mit ASP.NET 2.0 neu hingekommenen Skin-Files ermöglichen die Definition von Control Styles an einer zentralen Stelle. Fügt man dieser Datei ein Control hinzu, wird die Definition für alle verwendeten Controls der Applikation verwendet. Allerdings darf man an dieser Stelle tatsächlich auch nur Server-Controls hinzufügen, welche man an dem Attribut `runat="Server"` erkennt. Hat man dieses Attribut vergessen, zeigt Visual Studio oben genannte Fehlermeldung an.

Mehr Informationen zum Thema Skin-Files und ASP.NET Themes findet man unter:

http://quickstarts.asp.net/QuickStartv20/aspnet/doc/themes/default.aspx
http://www.microsoft.com/germany/MSDN/webcasts/library.aspx?id=118767541
http://msdn2.microsoft.com/de-de/library/wcyt4fxb(VS.80).aspx
http://www.15seconds.com/issue/040105.htm

2.4.9.Login.aspx: Cannot convert type

Grade wieder drüber gestolpert:

Benennt man eine Datei mit dem Namen `Login.aspx`, wird dazu in der Code-Beside Datei die Klasse `Login` angelegt. Was im Development Server von Visual Studio noch funktioniert, wird im IIS mit folgender Fehlermeldung quittiert:

Compiler Error Message:

```
CS0030: Cannot convert type 'ASP.login_aspx' to 'System.Web.UI.WebControls.Login'
```

Source Error:

```
Line 112: public login_aspx() {
Line 113: string[] dependencies;
Line 114: ((Login)(this)).AppRelativeVirtualPath = "~/login.aspx"
Line 115: if ((global::ASP.login_aspx.@__initialized == false)) {
Line 116: dependencies = new string[1];
```

Anscheinend besteht hier ein Namenskonflikt zwischen der Klasse `Login` im Namespace `System.Web.UI.WebControls` und der Klasse *Login* der eigentlichen Page.

Um dieses Problem zu lösen reicht es einfach die Klasse in der Code-Beside Datei umzubenennen. Im einfachsten Fall ergänzt man den Namen um einen Unterstrich.

```
public partial class _Login
```

In der aspx-Datei muss der Name ebenfalls angepasst werden

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Login.aspx.cs"
Inherits="_Login" %>
```

2.4.10.Von welcher Methode aufgerufen?

Für eine Logging Routine, welche bestimmte Ereignisse in einer Datenbank speichert, benötigte ich unter anderem den Namespace, die Klasse und den zugehörigen Methodennamen, von wo aus die Logging Routine aufgerufen wurde.

Die Informationen kann man mit Hilfe der Klasse `StackTrace` im `System.Diagnostics` Namespace auslesen werden.

```
System.Diagnostics.StackTrace stackTrace = new System.Diagnostics.StackTrace();
System.Diagnostics.StackFrame stackFrame = stackTrace.GetFrame(1);
System.Reflection.MethodBase methodBase = stackFrame.GetMethod();
```

Zunächst wird eine Instanz der Klasse `StackTrace` erstellt und mit Hilfe der Methode `GetFrame(1)` (beginnend bei 0) die Aufrufliste des aktuellen Threads

ausgelesen. Nun kann man mit `GetMethode()` eine Verbindung zu der Methode, welche die Logging Routine aufgerufen hat, herstellen.

```
string methodFullName = String.Format("{0}.{1}.{2}",
methodBase.DeclaringType.Namespace, methodBase.DeclaringType.Name,
methodBase.Name);
```

Über die Eigenschaften `Namespace` und *Name* des `DeclaringType` kann man, wie der Name schon sagt, den Namespace und den Klassennamen auslesen, während das Property `Name` des `methodBase` Objekts, den wirklichen Namen der Methode ausgibt.

2.4.11.User and Role Management in ASP.NET 2.0

Verwendet man die ASP.NET Membership und Role Provider sucht man evtl. eine Möglichkeit auf einfache Weise die User und Roles zu verwalten, ohne dieses direkt zu programmieren. Normalerweise kann dies über die ASP.NET Web Site Administration Tool erledigt werden, welches allerdings nach dem deployen der Website nicht mehr zur Verfügung steht.

Brennan Stehling hat sich die Mühe gemacht und Control programmiert, die genau für diese Aufgabe gedacht sind. Eine kurze Beschreibung und das Beispielprojekt findet man unter dieser URL.

Komischer weise funktioniert der Download des Zip-Archives nur mit FireFox, ansonsten erhält man eine defekte Datei.

Eine Anleitung um das Web Seite Administration Tool im Web zu hosten gibt es im Weblog von Alex.

2.4.12.Custom Controls per Web.config registrieren

Wie ich bereits in einem anderen Beitrag geschrieben hatte, ist es möglich User Controls per Web.Config zu registrieren, und diese somit auf allen Seiten zu nutzen.
Natürlich ist dies auch mit Custom Controls möglich. Auch hier muss im Node *Controls* mit Hilfe des `add` Tags das Control hinzugefügt und somit referenziert werden. Vorher sollte man dieses allerdings per *Add Reference* zum Web hinzugefügt haben.

```
<pages>
  <controls>
    <add assembly="MyCustomControl"
          namespace="MyCustomControl"
          tagPrefix="CC" />
  </controls>
</pages>
```

Natürlich steht jetzt auch wieder die Visual Studio IntelliSense für dieses Control zur Verfügung.

Abbildung 21: Custom Controls und IntelliSense

Den Download des Beispielprojekts findet man hier.

2.4.13.Login-Cookie des Community Server verwenden

Betreibt man neben dem Community Server eine weitere Anwendung, welche auch die ASP.NET Membership Controls nutzt, macht es Sinn das Login-Cookie in beiden Anwendungen zu nutzen. Der User muss sich somit nur einmal einloggend um in beiden Anwendungen arbeiten zu können.

Die Anwendungen selbst müssen innerhalb der gleichen Domain ausgeführt werden, z.B. www.domain.de und www.domain.de/forum

Weiterhin ist es nötig in beiden Web.Config folgende Einstellungen hinzuzufügen oder anzupassen, falls diese bereits vorhanden sind.

```
<machineKey
    validationKey="xxxx"
    decryptionKey="xxxx"
    validation="SHA1" />
```

Die Werte für die einzelnen Attribute können entweder direkt aus der `Web.Config` des Community Servers übernommen, oder selbst erstellt werden. Des Weiteren muss der Eintrag Authentication gleich sein

```
<authentication mode="Forms">
  <forms name=".CommunityServer"
        protection="All"
        timeout="60000"
        loginUrl="/default.aspx"
        slidingExpiration="true" />
</authentication>
```

Löscht man nun die Cookies im Browser und meldet sich erneut kann, wird das Cookie ebenfalls in der anderen Anwendung benutzt.

2.4.14.“Wirklich löschen?“ im GridView

Das GridView in Verbindung mit den DataSource Controls bietet von Haus aus die Möglichkeit Datensätze zu löschen. Allerdings werden die Datensätze ohne eine weitere Bestätigung des Users gelöscht. Für diesen Fall bietet sich ein kurzer JavaScript Dialog an.

Dafür ist es zunächst nötig die Spalten über den Befehl *Edit Columns* zu editieren

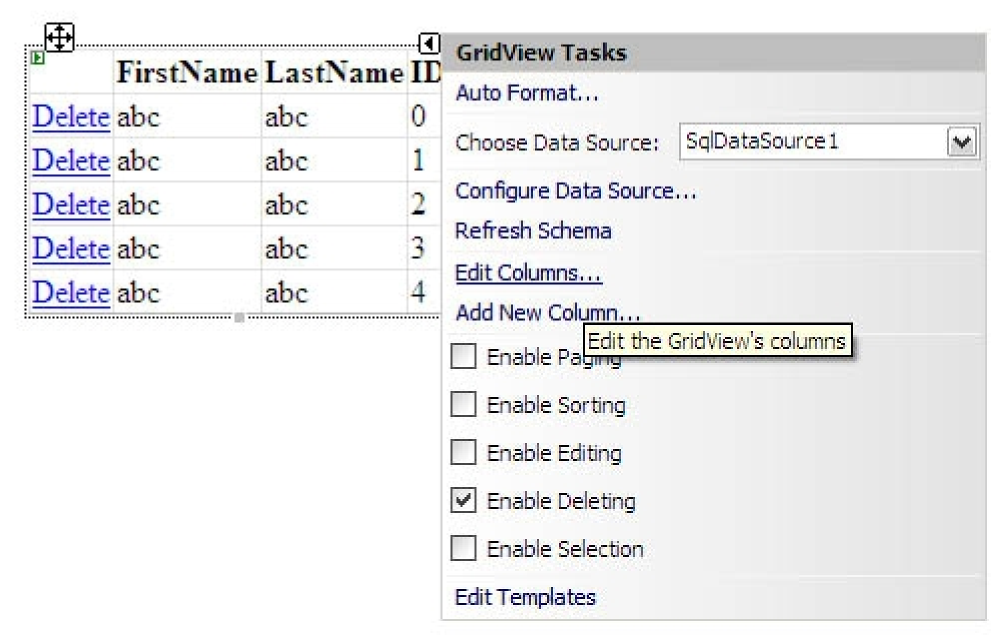


Abbildung 22: "Wirklich löschen" im GridView 1

Das Feld *Delete* muss anschließend in ein *TemplateField* konvertiert werden, welches mit einem Klick auf den Link *Convert this field into a TemplateField* erledigt ist.

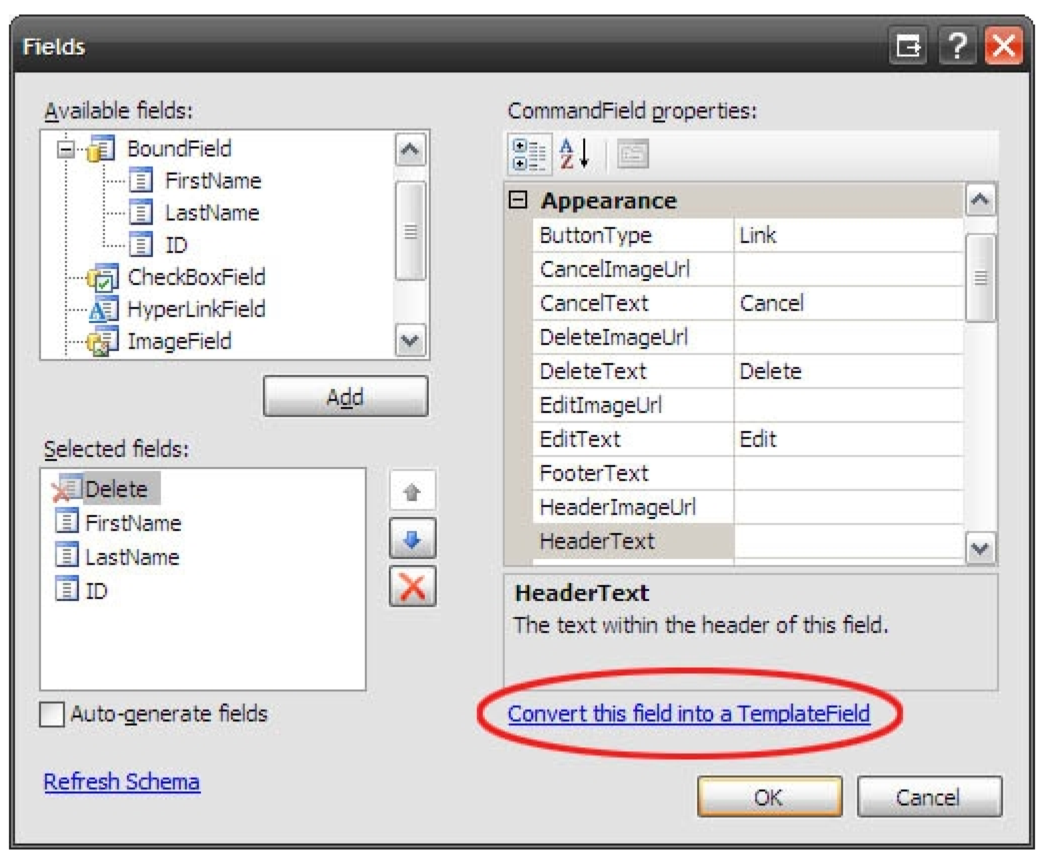


Abbildung 23: "Wirklich löschen" im GridView 2

Nach der Konvertierung steht ein vollwertiges LinkButton Control zur Verfügung. Das Attribut `OnClientClick` des Controls kann nun dazu benutzt werden um eine JavaScript Anweisung auszuführen, sobald der Button geklickt wird.

```
<asp:TemplateField ShowHeader="False">
    <ItemTemplate>
        <asp:LinkButton ID="LinkButton1" runat="server" CausesValidation="False"
            CommandName="Delete" OnClientClick="return confirm('Wirklich
löschen');"
            Text="Delete">
        </asp:LinkButton>
    </ItemTemplate>
</asp:TemplateField>
```

Die JS Methode `Confirm` öffnet eine Hinweis-Fenster und bietet die Buttons *OK* und *Abbrechen* an. Entsprechend des Klicks, liefert die Methode anschließend ein `true` oder `false` zurück.



Abbildung 24: "Wirklich löschen" im GridView 3

Der Rückgabewert wird nun per `return` Anweisung weitergereicht und somit die Verarbeitung des JavaScript zum Löschen des Datensatzes abgebrochen oder weitergeführt.

Den Download des Beispielprojekts gibt es hier.

2.4.15.Pop-Up per Response.Redirect()

Immer mal wieder taucht in diversen Foren die Frage auf: "Wie kann ich ein Pop-Up per `Response.Redirect()` öffnen". Um die Frage hinreichend zu beantworten, bedarf es einer weiteren Erklärung. Grundsätzlich wird in der Webforms-Entwicklung zwischen zweier Arten Code unterschieden. Code, der auf dem Client abgearbeitet wird, und Code, der auf dem Server abgearbeitet wird. Um dies zu verdeutlichen habe ich eine kleine Grafik angefertigt.

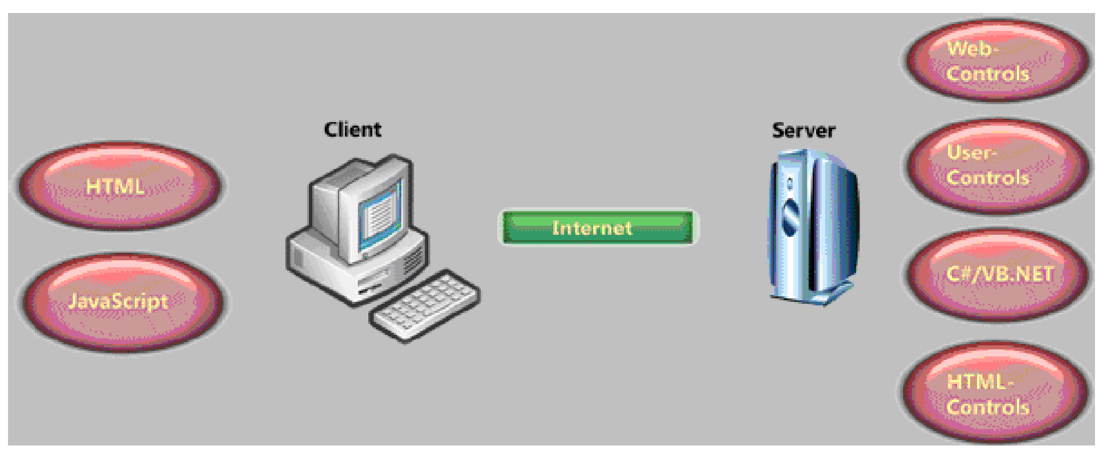


Abbildung 25: PopUp per Response.Redirect

Wie die Grafik zeigt, verarbeitet der Server div. Controls und C# bzw. VB.NET Code. Ist dieser Code verarbeitet, wird nichts anderes als reines HTML an den Client übertragen. Zu diesem Zeitpunkt ist die Arbeit des Servers für diese Anfrage beendet. Dem Client obliegt nun die Interpretation dieses Codes. Da das Öffnen des Popups eine Aufgabe des Clients ist, beantwortet sich die oben gestellt Frage von selbst. Es ist nicht möglich ein Popup per `Response.Redirect()` zu öffnen, da das `Redirect()` ohne Wissen des Clients serverseitig ausgeführt wird.

Allerdings ist es möglich per C# JavaScript-Code einzufügen, der dann wiederum vom Client beim Aufruf der Seite interpretiert wird. Hierfür bietet sich die Methode `RegisterStartupScript()` an.

```
Page.ClientScript.RegisterStartupScript(this.GetType(), "StartPopUp",
"window.open('http://www.asp.net');", true);
```

Eine genaue Beschreibung der Methode findet man in der MSDN.

2.4.16.Read-Only Datensätze im GridView, die Zweite

In diesem Eintrag beschrieb ich, wie man einzelne Zeilen im GridView auf Read-Only setzt, so dass diese nicht mehr bearbeitet oder gelöscht werden können. Anhand der Spalte `locked` wird entschieden, ob die Command Buttons funktionsfähig sind. Per E-Mail wurde ich nun mehrfach gefragt, ob es auch möglich ist Spalten auf Read-Only zu setzen und die Spalte `locked` nicht anzuzeigen. Hier nun die, zugegeben schnelle, Lösung.

```
protected void GridView1_RowDataBound(object sender, GridViewRowEventArgs e)
{
    if (e.Row.RowType == DataControlRowType.DataRow)
    {
        CheckBox chk = (CheckBox)e.Row.Cells[2].FindControl("CheckBox1");

        if(chk.Checked)
            e.Row.Cells[3].Enabled = false;
    }

    e.Row.Cells[2].Visible = false;
}
```

Die Spalte `locked` wird mit Hilfe der Eigenschaft `Visible` einfach ausgeblendet. Wichtig ist, dass diese Anweisung außerhalb der `if`-Anweisung ausgeführt wird, da ansonsten die Überschrift der Spalte weiterhin angezeigt wird.

2.4.17.Read-Only Datensätze im GridView

Die Aufgabe ist es, bestimmt Zeilen im GridView auf Read-Only zu setzen und somit die Möglichkeit diese zu löschen oder zu bearbeiten zu deaktivieren. In diesem Beispiel wird folgendes Datenbank-Layout verwendet.

Data

	Column Name	Data Type	Allow Nulls
🔑	ID	int	☐
	Name	varchar(50)	☐
	City	varchar(50)	☐
	locked	bit	☐
			☐

Abbildung 26: Readonly-Datensätze im GridView 1

Die Spalte `locked` zeigt an, ob die Spalte bearbeitet oder gelöscht werden darf. Zunächst muss das `SqlDataSource` Control konfiguriert werden. Neben dem Connection-String und dem Select Command, werden auch Update und Delete Command benötigt.

Anschließend kann das GridView konfiguriert und an das SqlDataSource Control gebunden werden. Bis auf das Feld `ID`, welches lediglich zur Identifizierung der Datensätze dient, werden alle anderen per Bound- bzw. TemplateField referenziert. Außerdem muss das Feld `locked` in Form einer CheckBox angezeigt werden, damit Datensätze manuell gesperrt werden können. In diesem Beispiel ist es nicht möglich einen Datensatz zu bearbeiten, sobald er gesperrt ist. Demnach ist es auch nicht möglich einen Datensatz wieder zu entsperren.

```
<asp:GridView ID="GridView1" runat="server" AllowSorting="True"
AutoGenerateColumns="False"
    DataSourceID="SqlDataSource1" Width="423px"
    DataKeyNames="ID"
    OnRowDataBound="GridView1_RowDataBound">
    <Columns>
        <asp:BoundField DataField="Name" HeaderText="Name"
SortExpression="Name" />
        <asp:BoundField DataField="City" HeaderText="City"
SortExpression="City" />

        <asp:TemplateField HeaderText="locked" SortExpression="locked">
            <EditItemTemplate>
                <asp:CheckBox ID="CheckBox1" runat="server"
                Checked='<%# Bind("locked") %>' />
            </EditItemTemplate>
            <ItemTemplate>
                <asp:CheckBox ID="CheckBox1" runat="server"
                Checked='<%# Bind("locked") %>' Enabled="false" />
            </ItemTemplate>
        </asp:TemplateField>
        <asp:CommandField ShowDeleteButton="True" ShowEditButton="True"/>
    </Columns>
</asp:GridView>
```

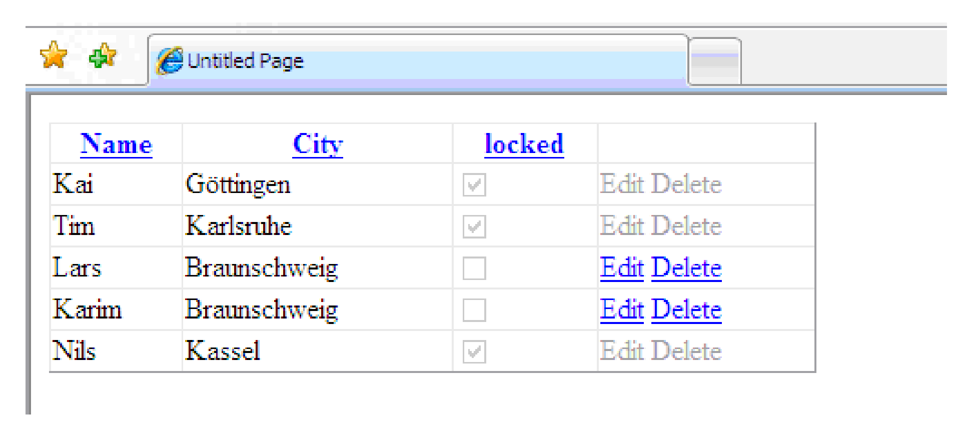
Wichtig ist nun das Attribut `OnRowDataBound`. Dieses gibt an welche Methode aufgerufen wird, sobald das Event `OnRowDataBound` vom GridView gestartet wird. Wie der Name schon sagt, wird dieser Event direkt nachdem ein Datensatz an das GridView gebunden wird aufgerufen. Aufgabe dieser Methode ist es nun zu überprüfen, ob der Datensatz gesperrt ist und ggf. die Command-Buttons zum Editieren oder Löschen zu deaktivieren.

```
protected void GridView1_RowDataBound(object sender, GridViewRowEventArgs e)
{
    if (e.Row.RowType == DataControlRowType.DataRow)
    {
        CheckBox chk = (CheckBox)e.Row.Cells[2].FindControl("CheckBox1");

        if(chk.Checked)
            e.Row.Cells[3].Enabled = false;
    }

}
```

Zunächst wird überprüft, ob es sich bei der aktuellen Zeile um eine Datensatz-Zeile handelt. Diese ist nötig, da beim Hinzufügen von Überschriften und Fußzeilen ebenfalls dieser Event aufgerufen wird. Anschließend sucht die Methode `FindControl()` die CheckBox anhand der definierten ID. Ist diese nun angehakt, wird die dritte Zelle der aktuellen Zeile des GridView deaktiviert. Das Ergebnis sind gesperrte Command-Buttons im GridView:



Name	City	locked	
Kai	Göttingen	☑	Edit Delete
Tim	Karlsruhe	☑	Edit Delete
Lars	Braunschweig	☐	Edit Delete
Karim	Braunschweig	☐	Edit Delete
Nils	Kassel	☑	Edit Delete

Abbildung 27: Readonly-Datensätze im GridView 2

Das Beispielprojekt kann hier heruntergeladen werden.

2.4.18.Dynamischer Meta-Refresh in MasterPage

Heute wurde in einem Forum die Frage gestellt, wie man eine HTML-Seite (bzw. Web Form) nach einer fest definierten Zeit aktualisieren kann. Hier bietet sich ein Meta-Refresh an, welcher im HEAD-Tag der Seite hinzugefügt werden muss.

```
<meta http-equiv="refresh" content="5;URL=http://www.asp.net">
```

Verwendet man allerdings eine MasterPage muss dieser HTML-Tag dynamisch hinzugefügt werden, ansonsten wäre er auf allen Seiten aktiv. Hierfür habe ich eine kleine Methode geschrieben.

```
private void AddMetaReferesh(int time, string url)
{
    HtmlGenericControl metaRefresh = new HtmlGenericControl("meta");
    metaRefresh.Attributes.Add("http-equiv", "refresh");
    metaRefresh.Attributes.Add("content", time + ";URL=" + url);
    Master.Page.Header.Controls.Add(metaRefresh);
}
```

Nachdem eine Instanz der `HtmlgenericControl` Klasse erstellt und die verschiedenen Attribute zugewiesen wurden, wird das Control dem Header der MasterPage hinzugefügt. Das Ergebnis ist ein dynamisch hinzugefügter Meta-Refresh.
Den Download des Beispielprojekts gibt es hier

2.4.19.Server.MapPath in Session_End ()

Möchte man die Methode `Server.MapPath` im `Session_End` Event der `global.asax` Datei verwenden, erhält man eine "*Object reference not set to an instance of an object" Exception*. Was zunächst ein wenig verwirrend ist, hat einen ganz einfachen Hintergrund. Die Klasse `HttpContext`, welche eine Instanz der Klasse `HttpServerUtility` und somit die Methode `MapPath` zurück gibt, ist null.

Der Grund hierfür ist, dass der `HttpContext` nur Request-bezogen zur Verfügung gestellt wird. Zum Zeitpunkt des Aufrufs von `Session_End`, ist die Session des Users (wie der Name schon sagt) bereits beendet und somit findet auch kein Request mehr statt.

Um die Funktionalität von `Server.MapPath` trotzdem abbilden zu können, kann man das Property `AppDomainAppPath` der `HttpRuntime` Klasse zur Hilfe nehmen.

```
private string MyMapPath(string file)
{
    return Path.Combine(HttpRuntime.AppDomainAppPath, file);
}
```

In Verbindung mit der `Path.Comine`, liefert diese Methode nun den kompletten Pfad zurück.

2.4.20.Dynamische Bilder per Generic Handler (*.ashx) anzeigen

Um dynamische Bilder in ASP.NET Seiten anzuzeigen verwendet man am besten einen Generic Hander *(*.ashx)*, anstatt eines Web Forms und die damit verbundenen Probleme. Diesen Tipp gab mir Thomas per Kommentar, bezugnehmend auf das Posting Formulare gegen SPAM schützen und Using themed css files requires a header control on the page

Grundsätzlich ist ein Web Form von der Klasse `System.Web.UI.Page` abgeleitet und ist somit dafür gedacht HTML, Webcontrols und Ähnliches anzuzeigen. Da die Anforderung aber einfach nur die Ausgabe eines Bildes per Stream ist, sind diese Dinge völlig unnötig. In diesem Moment kommt der Generic Handler ins Spiel.

Ein Generic Handler implementiert das `IHttpHandler` Interface und erlaubt es somit einen HTTP Handler zu schreiben, ohne ihn vorher kompilieren zu müssen. HTTP Handler werden normalerweise benutzt und auf eine vorher definierte Dateiendung reagieren zu können. Definiert man z.B. die Dateiendung *rss* in der `Web.Config`, leitet man die Verarbeitung an eine entsprechend Klasse, die das `IHttpHandler` Interface implementiert weiter. Diese Klasse übernimmt nun die Ausgabe des XML, welches man hinter der Endung *rss* erwarten würde.

In diesem Beispiel ist es aber lediglich gewollt ein Bild per Stream auszugeben. Fügt man eine *ashx* Datei dem Projekt hinzu, findet man eine Klasse mit der Methode `ProcessRequest` und der Eigenschaft `IsReusable` vor.

```
using System;
using System.Web;

public class CIImage : IHttpHandler {

    public void ProcessRequest (HttpContext context) {
        context.Response.ContentType = "text/plain";
        context.Response.Write("Hello World");
    }

    public bool IsReusable {
        get {
            return false;
        }
```

```
    }
}
```

Um den Zugriff auf den SessionState und somit den gespeicherten Text in der Session zu ermöglichen, muss die Generic Handler Klasse zunächst um das Interface `IRequiresSessionState` erweitert werden.

```
public class CIImage : IHttpHandler, IRequiresSessionState
```

Anschließend muss der Code zum Erstellen des Bildes in die Methode `ProcessRequest` kopiert und der `ContentType` entsprechend der Ausgabe verändert werden.

```
public void ProcessRequest (HttpContext context)
{
    context.Response.ContentType = "image/jpg";
    CaptchaImage.CaptchaImage ci = new
CaptchaImage.CaptchaImage(context.Session["CaptchaImageText"].ToString(), 200, 50,
"Century Schoolbook");
    context.Response.Clear();
    context.Response.ContentType = "image/jpeg";
    ci.Image.Save(context.Response.OutputStream,
System.Drawing.Imaging.ImageFormat.Jpeg);
    ci.Dispose();
}
```

Die Variable `context` vom Typ `HttpContext` ermöglicht die Ausgabe.

Ein Aufruf der Seite `CIImage.ashx` zeigt nun das Bild an, und birgt keine unerwarteten Probleme in sich. Den Download des erweiterten Beispielprojekts findet man hier.

2.4.21. Using themed css files requires a header control on the page

Mit dieser Fehlermeldung wird die Verarbeitung der ASP.NET Seite quittiert, sobald man das von mir beschriebene Captcha Control in eine Seite implementiert, die Themes verwendet.

ASP.NET erwartet den Head-Tag in Form eines Server-Controls, damit die entsprechen Theme Informationen eingebunden werden können:

```
<head runat="server">
```

Allerdings gibt es ASPX-Seiten in denen dieser Tag nicht vorhanden ist. In oben genanntem Beispiel übernimmt die Seite `CIImage.aspx` das Streaming des Captcha Images und somit würde dieser Tag zu Fehlern führen.
Um dieses Problem zu lösen hatte ich zunächst das Attribut `EnableTheming` im Page Header auf `false` gesetzt. Diese Einstellung bewirkt, dass die Seite während der Verarbeitung der Themes nicht berücksichtigt wird. Allerdings stört sich der ASP.NET Page Parser noch immer am fehlenden Tag und beendet den Aufruf mit derselben Fehlermeldung. Es führt also kein Weg am Tag vorbei.

Mit einem kleinen Trick ist es aber dennoch möglich den Page Parser zufrieden zustellen und eine korrekte Ausgabe zu gewährleisten:

```
<head runat="server" visible="false"/>
```

Das Attribut `visible` sorgt dafür das die Ausgabe ohne Tag erfolgt, während der Page Parser dieses vorfindet und somit die Verarbeitung nicht unterbricht.

2.4.22.Formulare gegen SPAM schützen

Um Formulare gegen SPAM-Einträge zu schützen bietet sich ein sog. Captcha an. Die Abkürzung Captcha steht für *"completely automated public Turing test to tell computers and humans apart".* Hinter diesem Wort verbirgt sich eine Technik, die mit Hilfe eines Bildes alphanumerische Zeichen darstellt.

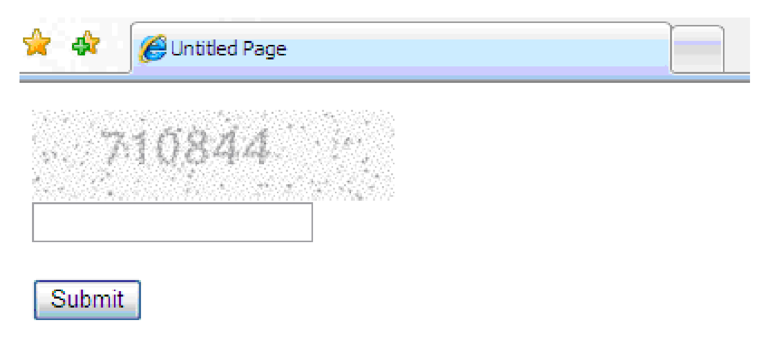


Abbildung 28: Captcha erstellen

Robots, die nun versuchen das Formular per Automatismus auszufüllen scheitern daran, die Zeichen abzulesen und in ein dafür vorgesehenes Formular-Feld einzutragen.

Neben div. kostenpflichtigen Controls findet man auf CodeProject.com eine fertige Klasse inkl. Beschreibung zur freien Verwendung. Die Implementierung ist nicht aufwendig und folgt einem einfachen Schema:

- *Generierte Zeichen in Session speichern*
- *Captcha Image erstellen und anzeigen*
- *Eingegebene Zeichen mit dem in der Session gespeicherten Wert vergleichen.*

Zunächst muss die Klasse in den `App_Code` Ordner des Webs kopiert und anschließend ein WebForm mit dem Namen `CIImage.aspx` hinzugefügt werden. Diese WebForm übernimmt das Streaming des Bildes, weshalb auch der gesamte Html-Code entfernt werden muss. In der `Page_Load()` Methode wird das Bild erstellt und an den `OutputStream` des `Response`-Objektes gesendet.

```
protected void Page_Load(object sender, EventArgs e)
{
    CaptchaImage ci = new CaptchaImage(Session["CaptchaImageText"].ToString(),
200, 50, "Century Schoolbook");
    Response.Clear();
    Response.ContentType = "image/jpeg";
```

```
    ci.Image.Save(this.Response.OutputStream, ImageFormat.Jpeg);
    ci.Dispose();
}
```

Die Seite `Default.aspx` übernimmt die Anzeige des Captcha-Images und auch die Generierung der angezeigten Zeichen. Ein einfacher `HtmlImage`-Tag wird verwendet um das Bild anzuzeigen. Die Überprüfung übernimmt ein `CustomValidator`, der die eingegebenen Zeichen mit denen der Session vergleicht.

```
<img src="CIImage.aspx" /><br />
<asp:TextBox ID="txtCaptchaText" runat="Server" /><br />
<asp:CustomValidator ID="valCaptcha" runat="server" ErrorMessage="Bitte geben Sie
den korrekten Code ein."
Display="dynamic" OnServerValidate="CheckCaptcha"/><br />
<asp:Button ID="cmdGo" runat="server" OnClick="cmdGo_Click" Text="Submit"
CausesValidation="true"/>
```

Die Zeichen werden mit Hilfe der `Random`-Klasse erstellt und anschließend in der Session gespeichert:

```
protected void Page_Load(object sender, EventArgs e)
{
    if(!IsPostBack)
        Session["CaptchaImageText"] = GenerateRandomCode();
}

private string GenerateRandomCode()
{
    Random random = new Random();

    string captchaCode = "";
    for (int i = 0; i < 6; i++)
        captchaCode = String.Concat(captchaCode, random.Next(10).ToString());
    return captchaCode;
}
```

Bei Formularen deren Daten sofort nach dem Submit sichtbar sind, wie z. B. für Gästebücher oder Kommentare für Weblog-Einträge, ist diese Absicherung durchaus sinnvoll. Auch die hier gezeigte, einfache Implementierung spricht dafür. Spätestens jedoch nachdem ein Robot mehrere Einträge verfasst hat, wird sie sogar absolut erforderlich.

Das Beispielprojekt findet man hier. Das Sourceforge-Beispiel inkl. Captcha-Control Klasse hier.

Wird dieses Beispiel in einer Anwendung mit Themes verwendet muss dies beachtet werden.

2.4.23.User Controls per Web.config registrieren

Wer oft mit User Controls arbeitet hat sich vielleicht schon darüber beschwert, die User Controls auf jeder Seite registrieren zu müssen. ASP.NET 2.0 bietet allerdings die Möglichkeit dies zentral in der `Web.config` zu erledigen.

Dazu muss die `Web.config` im Node `system.web` angepasst und folgender Code hinzugefügt werden:

```
<pages>
  <controls>
    <add tagPrefix="UC" tagName="World2" src="~/UC/MyUserControl1.ascx"/>
  </controls>
</pages>
```

Dieser sorgt dafür, dass das User Controls `MyUserControl1` nun für das gesamte Web registriert ist - Natürlich mit voller IntelliSense Unterstützung im Visual Studio.

Abbildung 29: Caching von Bildern verhindern

Ein kleines Beispielprojekt gibt es hier.

2.4.24.Caching von Bildern verhindern

In div. Foren stellen User häufiger die Frage, wie man das Caching von Bildern verhindern kann. Ein einfacher aber wirkungsvoller Trick ist es, an den Bildnamen eine Zufallszahl zu hängen. Meist reicht die Verwendung des Datums und der Uhrzeit:

```
image.ImageUrl = "Image/image.gif?" +
System.DateTime.Now.ToString("yyyMMddHHmmss");
```

Der Browser findet nun bei jedem Aufruf ein neuer Bildnamen vor und cached somit das Bild nicht.

2.4.25.XmlDataSource, GridView und DataFormatString

Die Darstellung einer XML – Datei ist mit Hilfe der `XmlDataSource` und des GridView kein Problem. Zumindest solange man nicht die Ausgabe der einzelnen Felder formatieren möchte. Hierbei stieß ich auf ein Problem, welches ich nur mit einem kleinen Workaround lösen konnte.

Zunächst der Aufbau der XML-Datei:

```
      <Data> <Item> <Number>0001</Number> <Name>Test
```

```
      1</Name> <Price>20.25</Price> <Date>12/11/1999</Date> </Item> <Item>
<Number>0002</Number> <Name>Test
      2</Name> <Price>1.75</Price> <Date>12/11/1999</Date> </Item> </Data>
```

Mit Hilfe des XmlDataSourceControls werden die Daten an das GridView gebunden.

```
<asp:GridView ID="grid" runat="server" DataSourceID="XmlDataSource1"

      AutoGenerateColumns="False"> <Columns> <asp:BoundField DataField="Number"
HeaderText="Number"

      SortExpression="Number" HtmlEncode="false"/> <asp:BoundField
DataField="Name" HeaderText="Name"

      SortExpression="Name" HtmlEncode="false"/> <asp:BoundField DataField="Price"
HeaderText="Price"

      SortExpression="Price" HtmlEncode="false" DataFormatString="{0:C2}" />
<asp:BoundField DataField="Date" HeaderText="Date"

      SortExpression="Date" HtmlEncode="false"
DataFormatString="{0:dd.MM.yyyy}" /> </Columns> </asp:GridView> <asp:XmlDataSource
ID="XmlDataSource1" runat="server" DataFile="~/App_Data/Data.xml"
TransformFile="~/App_Data/Data.xsl" EnableCaching="False"></asp:XmlDataSource>
```

Wie man sehen kann wird bei den Feldern `Price` und `Date` ein `DataFormatString` angewendet. Außerdem wird das Attribut `HtmlEncode` auf `false` gesetzt, um den bekannten Bug des GridView zu umgehen.

Allerdings zeigt der `DataFormatString` dennoch keine Wirkung, so dass die Ausgabe weiterhin unformatiert im Browser erscheint.

Number	Name	Price	Date
0001	Test 1	20.25	12/11/1999
0002	Test 2	1.75	12/11/1999
0003	Test 3	10.5	12/11/1999
0004	Test 4	15.5	12/11/1999
0005	Test 5	25	12/11/1999

Abbildung 30: GridView und XmlDataSource

Ein möglicher Workaround für diese Gegebenheit wäre die Verwendung des TemplateField und das manuelle Casten und Formatieren der Daten.

```
<asp:TemplateField> <ItemTemplate> <%#
      Convert.ToDateTime(Eval("Date")).ToShortDateString()%> </ItemTemplate>
</asp:TemplateField>
```

Die Ausgabe erscheint nun formatiert, allerdings ohne die Möglichkeit die Vorzüge der BoundFields zu nutzen.

2.4.26.Controls dynamisch hinzufügen

Immer mal wieder taucht in Foren die Frage auf, wie man Controls (wie z.B. die TextBox) dynamisch zu einer Seite hinzufügen kann. Mit Hilfe des PlaceHolder Controls ist dies ganz einfach möglich.

In diesem Beispiel soll der User die Anzahl der TextBox bestimmten können, die zur Seite hinzugefügt werden.

Zunächst muss das PlaceHolder Control, eine TextBox und ein Button auf der Seite platziert werden. In die TextBox trägt der User die Anzahl der TextBoxen ein, die hinzugefügt werden sollen. Der Event `OnClick` ruft die Methode `cmdAddControls_Click()`, liest die Anzahl aus und übergibt diese anschließend an die Methode `AddControls()`.

```
protected void cmdAddControls_Click(object sender,
    EventArgs e) { int numToAdd = Convert.ToInt32(txtAddNum.Text);
    AddControls(numToAdd); }
```

Die Methode `AddControls()` ist einfach aufgebaut. Innerhalb einer `for`-Schleife wird eine Instanz der TextBox-Klasse angelegt und diese dann der Control-Collection des PlaceHolders hinzugefügt.

```
private void AddControls(int number)
    { for (int i = 0; i < number;
    i++) { TextBox textbox = new TextBox(); textbox.Text = String.Format("Textbox
    {0}", i); plcControls.Controls.Add(textbox); plcControls.Controls.Add(new
LiteralControl("<br
    /><br />")); } }
```

Das Ergebnis dieser paar Zeilen sind dynamisch hinzugefügte TextBox-Controls. Allerdings müssen diese Controls nach jedem PostBack erneut hinzugefügt werden. Der Inhalt jeder TextBox wird allerdings vom ViewState gespeichert und bleibt somit erhalten.

Ein Tip am Rand: Das Control DynamicControlsPlaceholder von Denis Bauer speichert die hinzugefügten Controls auch über PostBacks hinweg.

Das Beispielprojekt gibt es hier.

2.4.27.DropDownList - Änderung der Auswahl durch JavaScript-Dialog bestätigen

Heute tauchte in einem Forum die Frage auf, ob es möglich ist die Änderung der Auswahl einer DropDownList nochmals durch den User per Dialog bestätigen zu lassen oder einen Hinweis auszugeben. Natürlich ist das möglich.

Sobald sich die Auswahl und somit auch der Index der DropDownList ändern, wird der clientseitige Event `onChange` ausgeführt. Die Lösung besteht also darin diesen

Event mit einer entsprechenden JavaScript-Methode zu belegen. Hierzu habe ich mir folgenden Code geschrieben:

```
private void RegisterDDLScript()
{
  string script = "if(!confirm('Sicher?')){document.getElementById('" +
   ddlList.ClientID+ "').selectedIndex = currentIndex;return false;}";
   ddlList.Attributes.Add("onchange", script);
}
```

Da der Event ausgeführt wird nachdem sich die Auswahl bereits geändert hat, muss man den vorherigen Index speichern. Das entsprechende Script hierfür registriert, in meinem Fall die Methode `RegisterStartUpScript()`:

```
private void RegisterStartUpScript()
{
   string script = "var currentIndex =
   document.getElementById('" + ddlList.ClientID + "').selectedIndex;";
   ClientScript.RegisterStartupScript(this.GetType(),     "ddlSelectedIndex",
script, true);
}
```

Das entsprechende Beispielprojekt findet man hier.

EDIT: Auf vielfachen Wunsch habe ich auch eine .NET Framework 1.1 Version des Beispielprojekts erstellt. Download

2.4.28.Cache löschen

Folgende kleine Methode habe ich heute geschrieben. Ihre Aufgabe ist es alle vorhandenen Cache-Items zu löschen.

```
private void DeleteAllCacheItems()
{
    HttpContext ctx = HttpContext.Current;
    IDictionaryEnumerator d = ctx.Cache.GetEnumerator();

    while(d.MoveNext())
    {
        ctx.Cache.Remove(d.Key.ToString());
    }
}
```

Brav wie ich bin, habe ich diese natürlich auch auf snippetcenter.org veröffentlicht .

2.4.29.Kopieren einer ASP.NET 1.1 Anwendung und VS 2003

Visual Studio .NET hat ermittelt, dass auf dem angegebenen Webserver nicht die ASP.NET, Version 1.1, ausgeführt wird. ASP.NET -Webanwendungen und -dienste können daher nicht ausgeführt werden.

Diese Fehlermeldung bekam ich heute angezeigt, nachdem eine Anwendung in einen anderen Ordner kopiert wurde. Diese verwirrende Fehlermeldung hat eine ganz einfache Ursache. Nachdem die Anwendung in einen anderen Ordner bzw. eine andere URL kopiert wurde, muss die `Datei <Projektname>.csproj.webinfo` angepasst werden.

Diese Datei wird geschrieben, sobald eine Anwendung erstellt wird:

```
<VisualStudioUNCWeb>
    <Web URLPath = "http://www.url.de/Anwendung.csproj" />
</VisualStudioUNCWeb>
```

Nachdem der Pfad angepasst wurde, kann man die Anwendung ohne Probleme öffnen. Verwendet man die Visual Studio Funktion "Projekt kopieren", muss man sich keine Gedanken um die Anpassung dieser Datei machen.

2.4.30.Controls anhand der ID rekursiv suchen

Schon vor längerer Zeit habe ich mir eine kleine Methode geschrieben, um Controls auf einer ASP.NET Seite per ID zu finden. Vielleicht hat der eine oder andere ja Verwendung dafür.

```
public static Control FindControlRecursive(Control root, string id) {
        if (root.ID == id) {
            return root;
        }

        foreach (Control c in root.Controls) {
            Control t = FindControlRecursive(c, id);
            if (t != null) {
                return t;
            }
        }

       return null;
    }
```

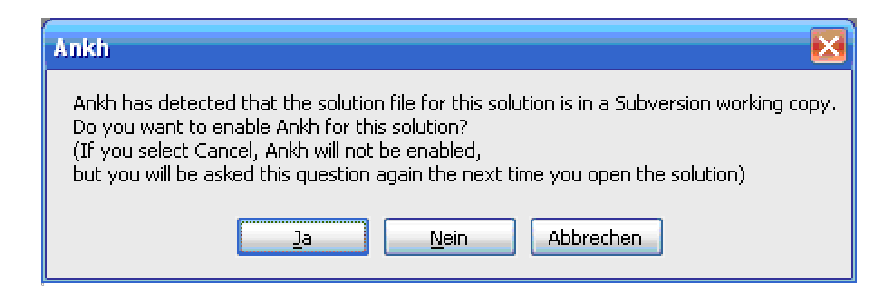
2.4.31.AnkhSVN, TortoiseSVN und ASP.NET

Verwendet man die TortoiseSVN ASP.NET Version, wird statt dem Standard Admin Ordner ".svn" der Ordner "_svn" erstellt. Visual Studio 2003 hat ein Problem mit Ordnern deren Namen mit einem Punkt beginnen, weshalb dieser Workaround gewählt wurde.
Nun kann es vorkommen, dass AnkhSVN bei Starten eines Projekts den Ordner ".svn" sucht, nicht findet und somit die Unterstützung für das Projekt deaktiviert. Damit der nach dem korrekten Ordner gesucht wird, reicht eine einfache Einstellung in der Configdatei von AnkhSVN. Die Datei findet man im Ordner:

```
C:\Dokumente und Einstellungen\<Benutzer>\Anwendungsdaten\AnkhSVN\ankhsvn.xml
```

In der Datei ersetzt man den Inhalt des Nodes `AdminDirectoryName` mit dem Wert "_svn". Per default war bei mir der Node noch auskommentiert. Nachdem die Änderungen durchgeführt wurden, reicht es wenn die Datei gespeichert und das Projekt geladen wird.

Hat alles geklappt, meldet sich Visual Studio bzw. AnkhSVN beim Starten mit folgendem Fenster:



2.4.32.Reservierte ASP.NET Projektnamen

Gibt man ein ASP.NET 1.1 Projekt z.B. den Namen "ValidationSummary" wird der Aufruf mit einer Fehlermeldung quittiert:

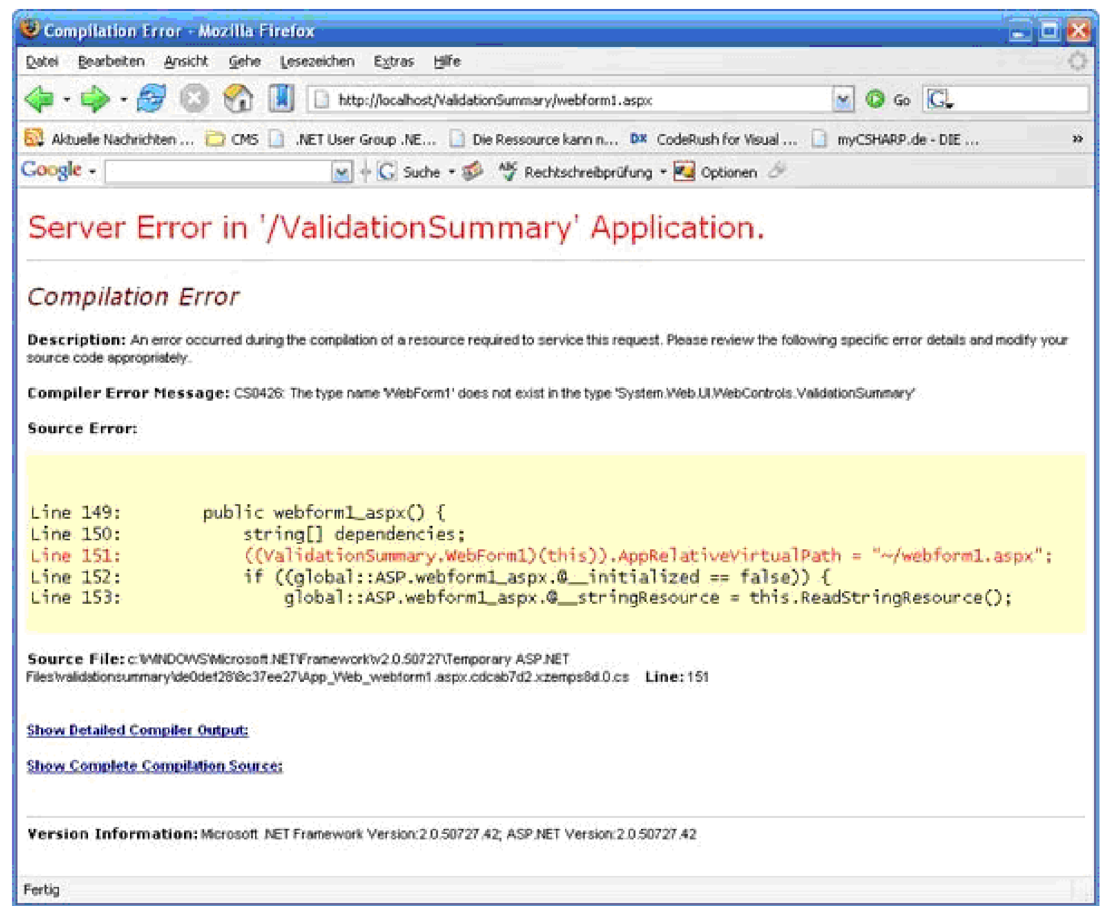


Abbildung 31: Reservierte ASP.NET Projektnamen

Der Grund für die Fehlermeldung ist klar, nur hätte Visual Studio die Namen der Projekte bzw. DLLs vielleicht schon beim Anlegen auf reservierte Wörter überprüfen sollen. Ich denke dabei an unbedarfte User, die Ihr erstes Projekt vielleicht "TextBox" nennen, gerade weil Sie mit diesem Control beginnen.

2.4.33.SiteMap menu with icons

Möchte man Icos in der Sitemap verwenden, sollte man sich folgende Lösung zu Gemüte führen:

http://weWeblogs.asp.net/bleroy/archive/2006/01/24/436350.aspx

2.4.34.Debug and Release Builds in ASP.NET 2.0

Wer sind nach ersten Experimenten mit ASP.NET 2.0 gefragt hat, wohin das /bin Verzeichnis und die Project dll verschwunden sind, sollte einen Blick auf den Artikel "Debug and Release Builds in ASP.NET 2.0" von K. Scott Allen unter folgender Adresse werfen:

http://odetocode.com/Weblogs/scott/archive/2005/11/15/2464.aspx

2.4.35.Maximum request length exceeded

Die Meldung wird ausgegeben, sobald man versucht eine Datei, die größer als 4096 KB ist per `HtmlInputFile` Control hoch zu laden. Um dieses Limit zu erhöhen überschreibt (bzw. ergänzt) man folgenden Wert in der `Web.Config` oder `Maschine.Config`:

```
<httpRuntime maxRequestLength="8192" />
```

`maxRequestLength` steht natürlich für den Maximalwert in KB.

Die Knowledge Base Artikel findet man dazu unter:
http://support.microsoft.com/default.aspx?scid=kb;EN-US;295626.

2.4.36.Session lost, nachdem ein Verzeichnis umbenannt wurde

Anscheinend verliert ASP.NET nachdem ein Verzeichnis umbenannt wurde die komplette Session. Diesen Fehler kann ich jederzeit reproduzieren und wurde mir auch von Olaf Lüder in microsoft.public.de.german.entwickler.dotnet.asp bestätigt. Dieser Fehler tritt auch auf, wenn leere Verzeichnisse umbenannt werden.

Folgender Workaround:

1. Neues Verzeichnis anlegen
2. Dateien kopieren
3. Altes Verzeichnis löschen.

Hierzu habe ich die Funktion `CopyFiles()` geschrieben:

```
public static void CopyFiles(string strSourceFolder,string
strDestinationFolder,string strRestrictions) {
     //Kopiert alles Dateien aus einem Verzeichnis in das andere.
     //Es können auch Einschränkungen (z.B. *.aspx) übergeben werden.

     if( strDestinationFolder.Substring(strDestinationFolder.Length-1,1) != "\\")
strDestinationFolder += "\\";

     DirectoryInfo dir = new DirectoryInfo(strSourceFolder);
     FileInfo[] files = dir.GetFiles(strRestrictions);
     foreach(FileInfo file in files) {
```

```
            file.CopyTo(strDestinationFolder + file.Name,true);
        }
    }
```

2.4.37.ASP.NET: HttpWebRequest und Timeout-Problem

Bis dato habe ich ja noch nicht viel mit ASP.NET gemacht und muss daher auf die eine oder andere Tücke erst kommen. Da ich mich beruflich nun verstärkt damit beschäftigen muss, wird wohl meine ASP.NET Kategorie ein wenig aufgefüllt werden.

Nun jedoch zum eigentlichen Thema: Wer via ASP.NET WebRequests aufbaut kann eventuell auf das Problem eines Timeouts stoßen. Meist beim zweiten Aufruf erfolgt ein Timeout. Des Rätsels Lösung liegt darin, die dazugehörigen Response-Objekte zu schließen. Vergisst man ein `Response.Close()` kommt es eben zum genannten Timeout-Problem. Also immer brav darauf Acht geben :)

2.4.38.Probleme mit WebUIValidation.js

Wer nach dem Deployment einer ASP.NET Anwendung oder bei der Entwicklung ein Problem mit der Datei WebUIValidation.js hat (es erscheint eine wunderschöne Fehlermeldung, dass diese Datei nicht vorhanden ist oder aber nur nicht verwendet werden kann), dem kann einfach geholfen werden.

Dazu einfach die Visual Studio 2005 Eingabeaufforderung öffnen und den Befehl

```
aspnet_regiis -c
```

eingeben. Ab sofort sollte wieder alles ganz normal funktionieren.

2.4.39.Advanced Captcha in ASP.NET

In der Diskussion Captcha-Ablösung wurde nach Möglichkeiten gesucht, herkömmliches Captcha abzulösen. Thomas Bandt löst dies durch eine einfache Rechenaufgabe, welche von den aktuellen Bots meist nicht gemeistert werden kann.

Ich habe auf Basis des CodeProject-Artikels CAPTCHA Image eine (zumindest für mich neue) Variante entwickelt. Zum Schluss sieht dies so aus:

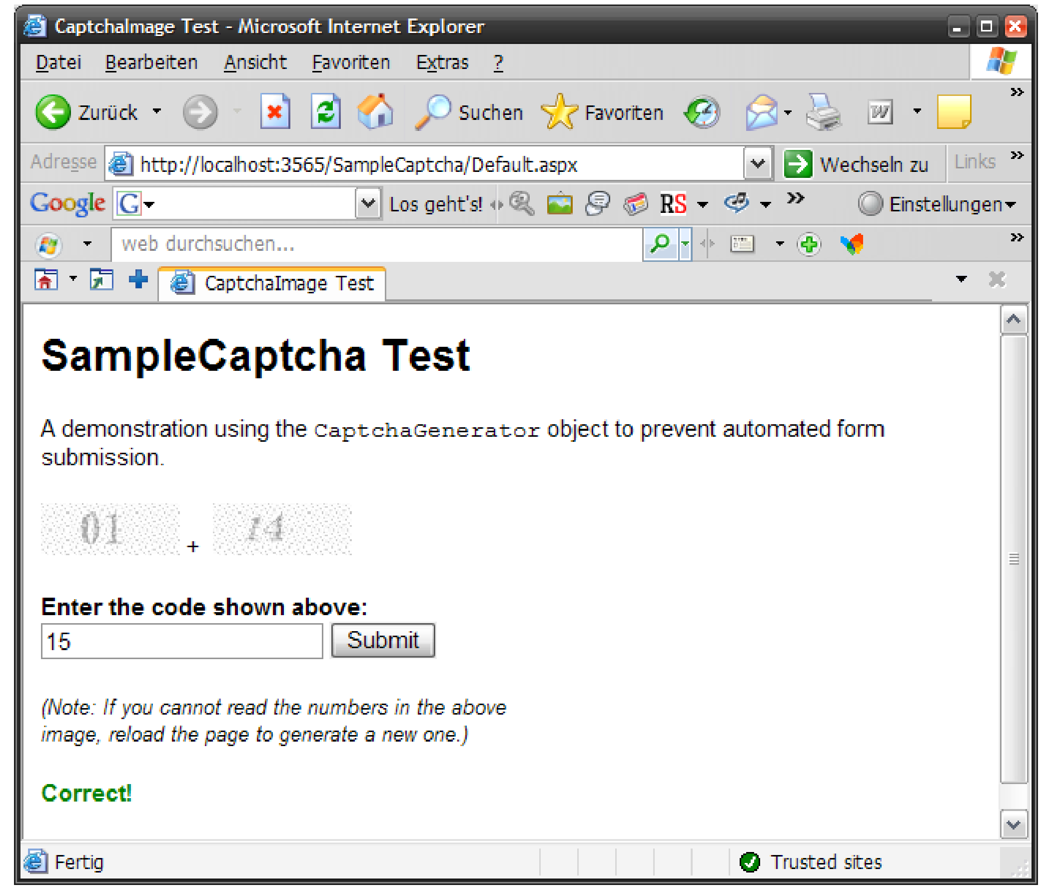


Abbildung 32: Captcha-Variante

Der Schwierigkeit kann hierbei noch erhöht werden, indem zwischen die Zahlen Buchstaben oder andere Zeichen eingefügt werden, die der intelligente Benutzer auslassen muss.

Wer sich dafür interessiert, kann das Projekt SampleCaptcha herunterladen.

2.4.40. Advanced Captcha in ASP.NET: Eine weitere Variante

So, hier habe ich noch eine Variante des bereits vorgestellten Captchas. Auch dafür sollte es aktuell noch keine entsprechenden Implementierungen in den diversen Bots geben.

Sieht dann so aus:

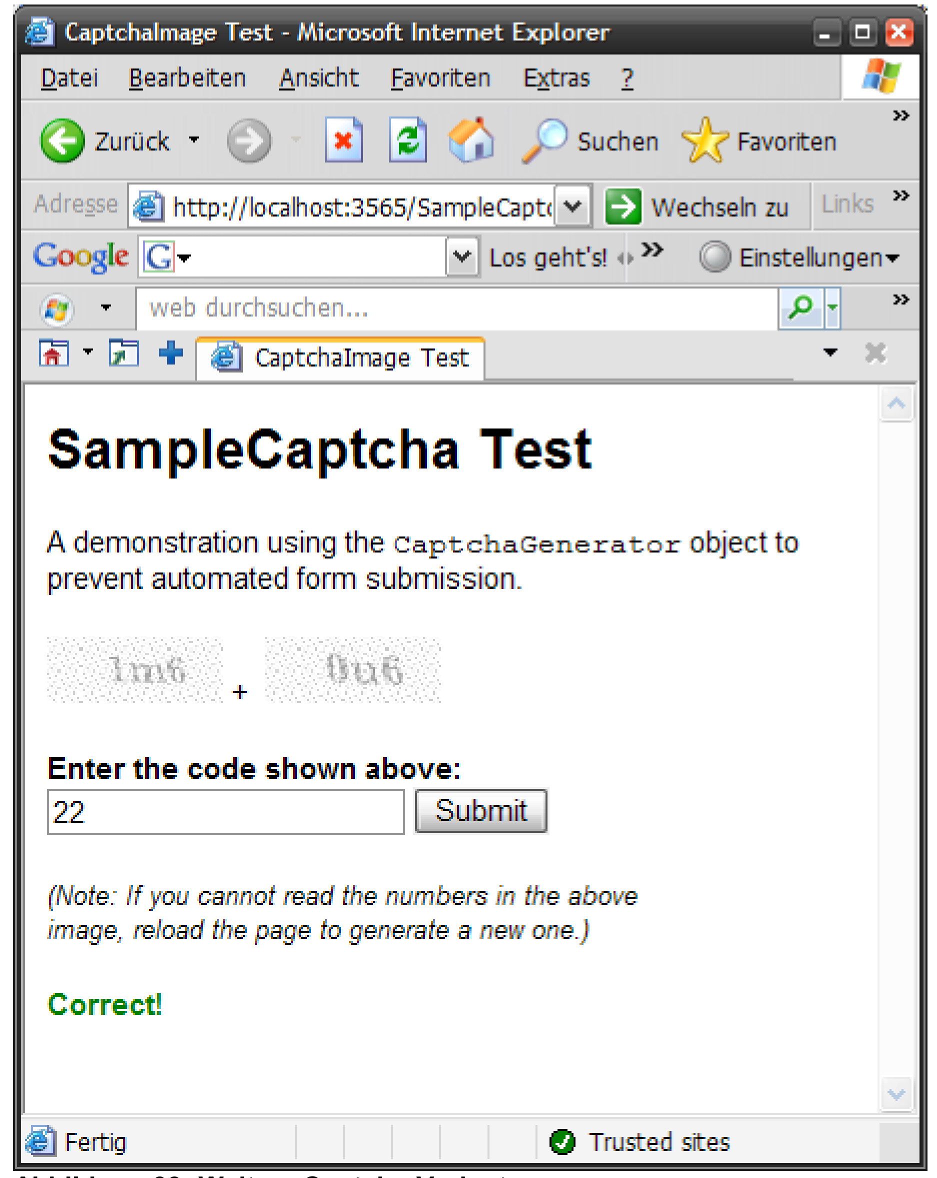


Abbildung 33: Weitere Captcha-Variante

[Captcha Sample 2 Download]

2.4.41.JavaScript Alert von CodeBehind-File aufrufen

In vielen Foren wird oft die Frage gestellt, wie denn aus beispielsweise C# heraus eine `MessageBox` im Web angezeigt werden kann.

Folgende Klasse zeigt die Verwendung:

```
public static class Alert
{
    public static void Show(string message)
    {
        string repMessage = message.Replace("'", "'");
        string script = "";
```

```
        Page page = HttpContext.Current.CurrentHandler as Page;

        if (page != null
         &&
         !page.ClientScript.IsClientScriptBlockRegistered("alert"))
        {
            page.ClientScript.RegisterClientScriptBlock(
                typeof(Alert),
                "alert", script);
        }
    }
}
```

Unsere Funktionalität kann nun von jeder beliebigen Stelle mit

```
Alert.Show("Dies ist ein Test");
```

aufgerufen werden.

2.4.42.ASP.NET AJAX Linksammlung

Hier ein paar Links zum Thema **ASP.NET AJAX**:

ASP.NET AJAX Download
ASP.NET AJAX Online Dokumentation
ASP.NET AJAX Video Tutorials
AJAX Control Toolkit
AJAX Control Toolkit Demos

Weiters gibt es zu diesem Thema noch die ASP.NET 2.0 AJAX Futures January CTP. Hierfür muss jedoch müssen jedoch ASP.NET AJAX 1.0 installiert sein. In der Futures CTP befinden sich in Entwicklung befindliche Funktionalitäten.

Zu guter Letzt: Die AJAX Cheat Sheets - eine übersichtliche Darstellung der Klassen und Funktionen.

2.4.43.HTML Seiten mit ASP.NET Cachen

Bei großem Useraufkommen und/oder auf Seiten die nicht immer 100%ig aktuell sein müssen, bietet es sich an, diese Seiten zu cachen. Dies bedeutet, dass die jeweilige Seite nicht bei jedem Aufruf neu gerendert werden muss.

Hier nun ein einfaches Beispiel für das Cachen von ASP.NET Pages. Die Ausgangslage bietet folgende Seite:

```
< %@ Page Language="C#" AutoEventWireup="true"
CodeFile="Default.aspx.cs" Inherits="_Default" % >
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Cache Demo</title>
</head>
<body>
    Uhrzeit: <% = DateTime.Now.ToLongTimeString() %>
</body>
</html>
```

Wird diese Seite gestartet und mehrmals durch **Reload** aktualisiert ist schön zu sehen, dass stets die aktuelle Uhrzeit angezeigt wird.
Mit der Output-Direktive kann nun ein einfaches Caching aktiviert werden. Der Aufbau der Direktive sieht wie folgt aus:

```
<%@ OutputCache Duration="10" VaryByParam=None %>
```

Hier nun das gesamte Beispiel:

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeFile="Default.aspx.cs" Inherits="_Default"%>
<%@ OutputCache Duration="10" VaryByParam=None %>

<!DOCTYPE
html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Cache Demo</title>
<</head>
<body>
    Uhrzeit: <% = DateTime.Now.ToLongTimeString() %>
</body>
</html>
```

Bei einer Ausführung dieser Seite und dem Betätigen von **Reload** ist schön zu sehen, dass der Seiteninhalt ca. alle 10 Sekunden aktualisiert wird.

2.4.44.ASP.NET AJAX aktualisierte Dokumentation und Videos

Das ASP.NET AJAX Team hat die Dokumentation zu ASP.NET AJAX 1.0 aktualisiert. Neben einigen Korrekturen und zusätzlichen Anmerkungen, gibt es neue Beispiele und weitere Code Snippets.

Die Dokumentation wird zum Download angeboten, oder kann direkt online gelesen werden.
Außerdem gibt es zwei neue Videos der Serie "How Do I?" with ASP.NET AJAX.

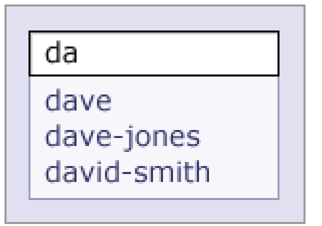


Abbildung 34: Ajax Auto-Complete

[#32 | How Do I: Use the ASP.NET AJAX AutoComplete Control]

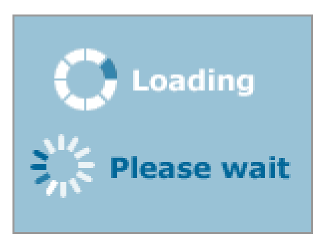


Abbildung 35: Ajax Progress

[#33 | How Do I: Use the ASP.NET AJAX UpdateProgress Control?]

2.4.45.A potentially dangerous Request.Form value was detected

Tritt die oben genannte Fehlermeldung auf, können gerade ASP.NET Einsteiger im ersten Moment nicht viel mit damit anfangen. Aber auch erfahrene Entwickler wählen häufig den einfachen und schnellen Weg dieser Fehlermeldung, oder besser, diesen Sicherheitsmechanismus zu umgehen.

ASP.NET überprüft Usereingaben, die per PostBack (*POST*) oder per Parameter im Link (*GET*) übermittelt werden. Werden in diesen Daten potentiell gefährliche Inhalte entdeckt, wird dies in Form eines Fehlers gemeldet. Zu den potentiell gefährlichen Inhalten gehören z.B. JavaScripts, mit denen man Cross-Site Scripting Angriffe durchführen könnte. Ausprobiert werden kann dies mit einer einfachen Textbox und der anschließenden Ausgabe des eingegebenen Inhaltes.

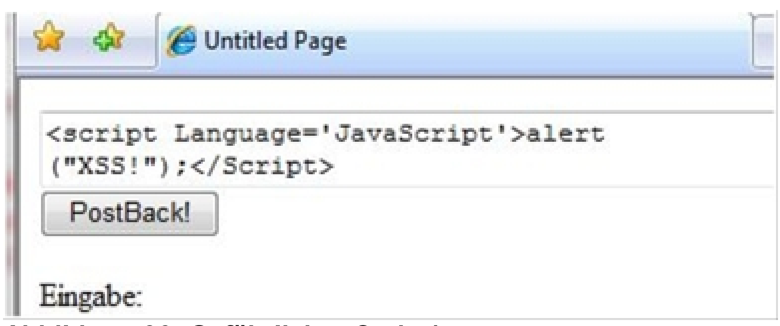


Abbildung 36: Gefährlicher Code 1

Ist die Überprüfung seitens ASP.NET deaktiviert, wird die Eingabe des Users direkt ausgeführt, sodass ein JavaScript-Alert mit dem Hinweis "XSS!" erscheint.

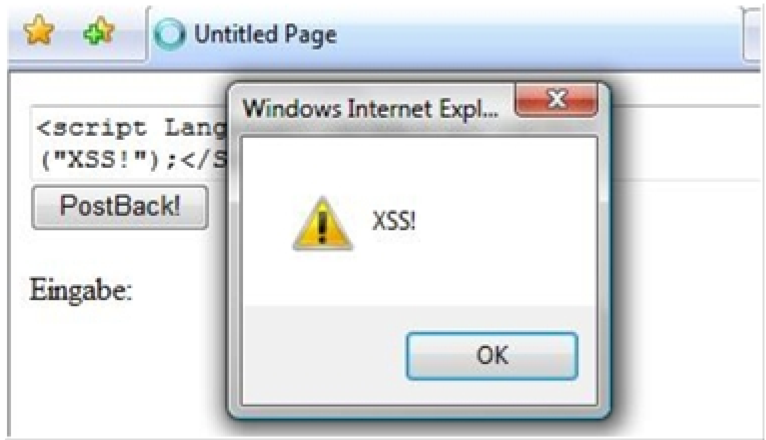


Abbildung 37: Gefährlicher Code 2

In machen Fällen ist die Eingabe von HTML oder JavaScript vom User aber ausdrücklich erwünscht. ASP.NET bietet an zwei Stellen die Möglichkeit die Überprüfung zu deaktivieren. Für die gesamte Applikation kann dies in der `Web.Config` konfiguriert werden.

```
<system.web>
  <pages validateRequest="false"/>
</system.web>
```

Soll die Überprüfung nur für eine Seite deaktiviert werden, wird dies über die gleichnamige Eigenschaft im `@Page-Header` erledigt.

```
<%@ Page Language="C#" AutoEventWireup="true"  CodeFile="Default.aspx.cs"
Inherits="_Default" ValidateRequest="false" %>
```

Die erste Möglichkeit sollte nur angewendet werden wenn die Anwendung in einer sicheren Umgebung läuft, sodass solche Eingaben oder Angriffe von vorne heraus ausgeschlossen werden können. Ist dies nicht der Fall muss sichergestellt werden das die Eingaben des Users für die Ausgabe umgewandelt ausgegeben werden. Das .NET Framework bietet hierfür die Methode `HtmlEncode()` an. Diese Methode wandelt nicht Html-Konforme Zeichen um, sodass diese korrekt im Browser angezeigt werden. Wird diese Methode im oberen Beispiel verwendet, sieht die Ausgabe wie folgt aus.

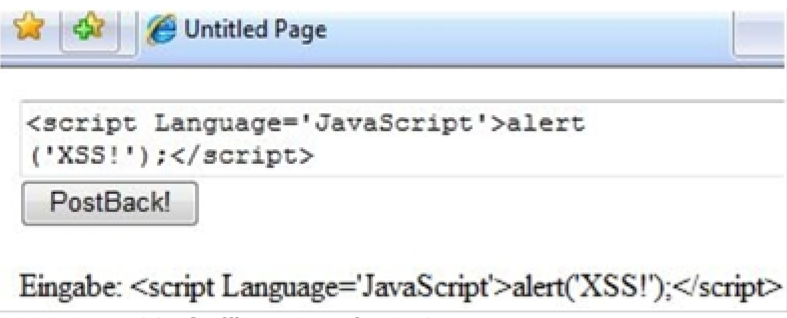


Abbildung 38: Gefährlicher Code 3

Mehr Informationen zu diesem Thema gibt es unter folgenden Links:

http://www.asp.net/faq/RequestValidation.aspx

http://www.microsoft.com/technet/archive/security/news/exsumcs.mspx?mfr=true
http://de.wikipedia.org/wiki/Cross-Site_Scripting

Das Beispielprojekt kann hier runtergeladen werden.

2.4.46. „Zuletzt geändert am“ automatisch anzeigen

Heute wurde ich per E-Mail gefragt ob es möglich ist das Datum und die Uhrzeit anzuzeigen, an dem die aktuelle ASPX-Datei das letzte Mal verändert wurde. In diesem Beispiel wird ein einfaches Custom-Control diese Aufgabe erledigen. Natürlich ist dies nicht der Weisheit letzter Schluss. Man kann dieses kleine Beispiel jedoch sehr gut erweitern und seinen Bedürfnissen anpassen.

Ein Custom-Control wird über ein Class-Library Projekt erstellt.

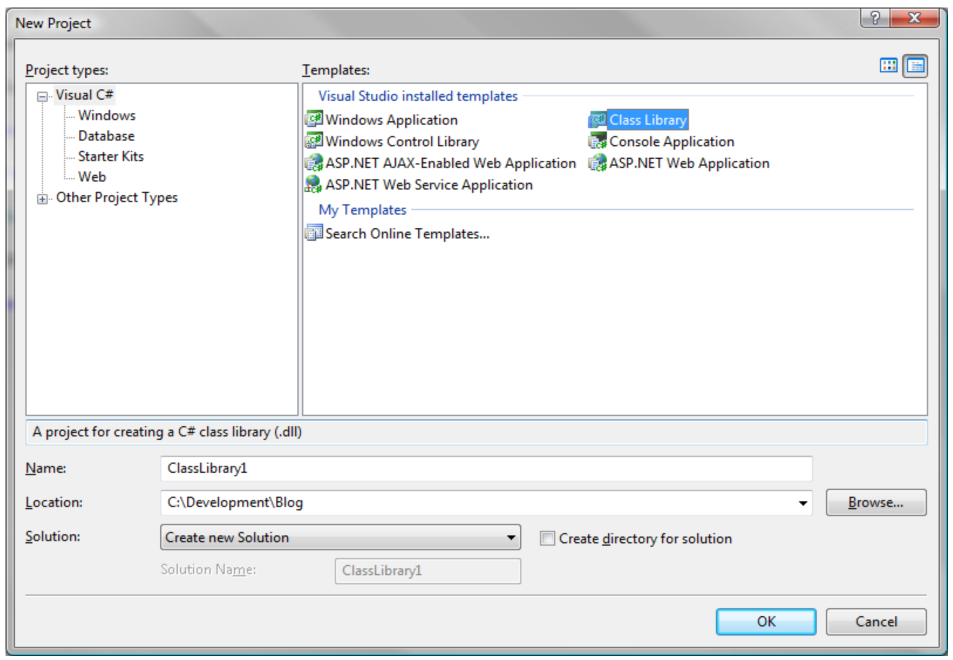


Abbildung 39: Custom Control

In diesem Fall wird die neue Klasse von einer vorhandenen abgeleitet, sodass man die grundlegende Funktionen nicht selbst implementieren muss. Da die Ausgabe nicht sehr umfangreich ist, bietet sich das Literal-Control an.

```
public class LastWriteTime : System.Web.UI.WebControls.Literal
{
    protected override void Render(System.Web.UI.HtmlTextWriter writer)
    {
        FileInfo fi = new FileInfo(HttpContext.Current.Request.PhysicalPath);

        writer.Write(fi.LastWriteTime);

        base.Render(writer);
    }
}
```

Wie man sehen kann, wird die Methode Render überschrieben bzw. um den nötigen Code ergänzt. Die Eigenschaft PhysicalPath beinhaltet den vollständigen physischen Pfad der aktuellen ASPX-Datei welcher nötig ist, um eine Instanz der Klasse `FileInfo` zu erstellen. Über die Eingenschaft `LastAccessTime` ist es möglich das Datum und die Uhrzeit auszulesen an dem die Datei - wie der Name schon sagt - zuletzt geschrieben wurde. Die Methode `Write` des `writer`-Objekts gibt diese Info aus.

Im Web-Projekt selbst kann man das Custom-Control per `Web.Config` hinzufügen. Bereits an [anderer Stelle] hatte ich beschrieben wie dies möglich ist. Der Vollständigkeit halber folgend nochmal der Eintrag in der `Web.Config`.

```
<pages>
  <controls>
    <add assembly="BlaBlubWeblogUtils" namespace="BlaBlubWeblogUtils"
tagPrefix="CC"/>
  </controls>
</pages>
```

Nun ist das Custom-Control auf jeder Seite verfügbar und kann mit dem entsprechenden Tag eingebunden werden.

Die Datei wurde zuletzt am `<CC:LastWriteTime runat="Server" />` Uhr verändert.

Den Download der beiden Projekte findet man [hier].

2.4.47.ViewState Helper

Der ViewState Helper von Binary Fortress Software erleichtert die Überprüfung der Größe des ViewStates. Ist das Programm gestartet, protokolliert es die aufgerufenen Seiten und gibt die Größe des ViewState aus. Neben diversen anderen Informationen ist es außerdem möglich das ViewState zu dekodieren und somit anzuschauen.

Abbildung 40: ViewState Helper

Das Tool ist kostenlos und benötigt lediglich das .NET Framework in der Version 2.0 Mehr Infos und den Download findet man unter http://www.binaryfortress.com/aspnet-viewstate-helper/.

2.4.48.Caching, Membership, Roles, Profiles, Web-Architektur und AJAX

Daniel Walzenbach hat in seinem WeWeblog einige Interessante Links zum Thema ASP.NET geposted, die in jeden Fall ein Blick wert sind.

3-stufige Web-Architektur für ASP.NET (VB.NET oder C#)
A Multipart Series on ASP.NET 2.0's Membership, Roles, and Profile
Caching Tutorial for Web Authors and Webmasters

Außerdem gibt es von Microsoft Learning einen kostenlosen 2h Online-Kurs zum Thema ASP.NET AJAX 1.0

https://www.microsoftelearning.com/eLearning/courseDetail.aspx?courseId=73568

2.4.49.Fehlerbehandlung per Http Module

Mit einem HTTP Module ist es möglich Fehler in Web-Anwendungen zu protokollieren, ohne den vorhandenen Code der Anwendung zu verändern. Die Aufgabe eines solchen Modules sollte es sein, dem User eine freundliche Meldung anzuzeigen, zusätzlich aber den vorhandenen Zustand festzuhalten und an den Entwickler zu verschicken oder in einer Datei zu speichern. In diesem Fall werden die Informationen per E-Mail verschickt.

Beginnt man mit der Programmierung eines HTTP Modules muss zunächst eine Klasse erstellt werden, die das Interface `IHttpModule` implementiert.

```
public class ErrorHandler : IHttpModule
{
    #region IHttpModule Members

    public void Dispose()
    {
        throw new Exception("The method or operation is not implemented.");
    }

    public void Init(HttpApplication context)
    {
        throw new Exception("The method or operation is not implemented.");
    }

    #endregion
}
```

Visual Studio erstellt auf Wunsch die Methoden Dispose() und Init(), die durch das Interface vorgegeben sind. In der Methode OnInit muss nun ein Event-Handler für den Error Event registriert werden.

```
public void Init(HttpApplication context)
{
    context.Error += new System.EventHandler(Error);
}

private void Error(object o, EventArgs e)
{

}
```

In dieser Methode kann nun die weitere Verarbeitung vorgenommen werden. Zunächst ist es nötig, den aktuellen Request und den ausgelösten Fehler einzulesen. Hierfür liefert das Property Current der HttpContext Klasse den aktuellen HttpContext zurück, welcher dann die nötigen Objektinstanzen liefert.

```
HttpContext context = HttpContext.Current;
HttpResponse response = context.Response;
HttpRequest request = context.Request;

Exception exception = context.Server.GetLastError();
```

Das Exception-Objekt beinhaltet nun die Informationen über den aufgetretenen Fehler. In diesem Beispiel wird einfach der gesamte Inhalt der Exception per E-Mail verschickt. Wie aber bereits angesprochen, wären auch andere Vorgehensweisen denkbar, z.B. eine Persistierung des Fehlers in eine Datei oder Datenbank. So ist es möglich Statistiken über aufgetretene Fehler zu erstellen und auszuwerten. Anschließend sollte dem User eine freundliche Fehlermeldung präsentiert werden, sodass er nicht mit der rohen Fehlermeldung konfrontiert wird.

Der ErrorHandler wird in der Web-Anwendung über die Web.Config registriert. Hierfür ist der Abschnitt httpModules vorgesehen.

```
<httpModules>
  <add type="WebAppUtils.ErrorHandler, WebAppUtils" name="WebAppUtils"/>
</httpModules>
```

Anschließend übernimmt die Bearbeitung des Fehlers das entwickelte Http Module.

Den Download der Beispielprojekte findet man hier.

2.4.50. Verschiedene Programmiersprachen im App_Code Ordner verwenden

Im App_Code Ordner werden Klassen abgelegt, auf die jede ASP.NET Page der Applikation zugreifen kann. Angenommen man möchte nun eine VB.NET Datei in

einem App_Code Ordner verwenden, der sonst nur C# Dateien enthält. Visual Studio straft diesen Versuch mit einer mehr oder weniger langen Fehlermeldung ab.

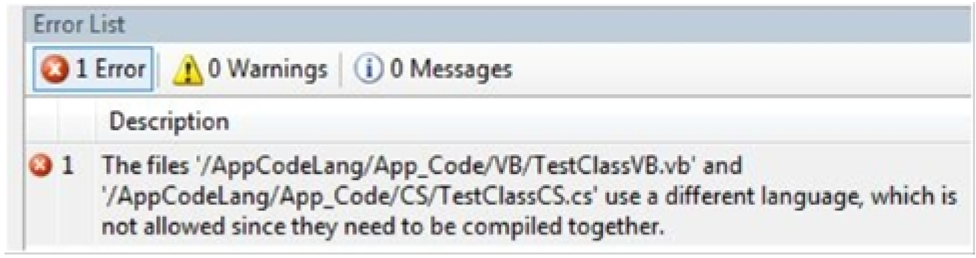


Abbildung 41: ASP.NET - Unterschiedliche Programmiersprachen

Der Grund für die Fehlermeldung ist der Umstand, dass der gesamte Inhalt des App_Code Ordner als Assembly kompiliert wird. Innerhalb dieses Assemblies ist nur die jeweils ausgewählte Sprache zulässig. Nun hat man die Möglichkeit den gesamten Inhalt der VB.NET Datei in C# Code zu portieren. Bei sehr umfangreichem Code kann dies schnell in sehr viel Arbeit ausarten. Eine zweite Möglichkeit wäre den Code per Hand in ein Assembly auszulagern und dies in der Applikation zu referenzieren. ASP.NET selbst bietet allerdings eine weitere Möglichkeit an, die das Problem sehr schnell löst.

```
<compilation debug="true">
  <codeSubDirectories>
    <add directoryName="VB"/>
  </codeSubDirectories>
</compilation>
```

Diese Einstellung gibt dem Compiler die Anweisung, dass der Ordner *VB* des App_Code Verzeichnisses in ein separates Assembly ausgelagert werden soll. Anschließend startet die Anwendung ohne die zu Anfang genannte Fehlermeldung.

Den Download des Beispielprojekts findet man hier.

2.4.51.„machineKey“ Generator

Auf ASP.NETResources.com findet man einen Generator um schnell und unkompliziert Keys für die Überprüfung und Verschlüsselung bzw. Entschlüsselung des ViewState zu generieren. Der generierte Key kann dann einfach in die Web.Config übernommen werden.

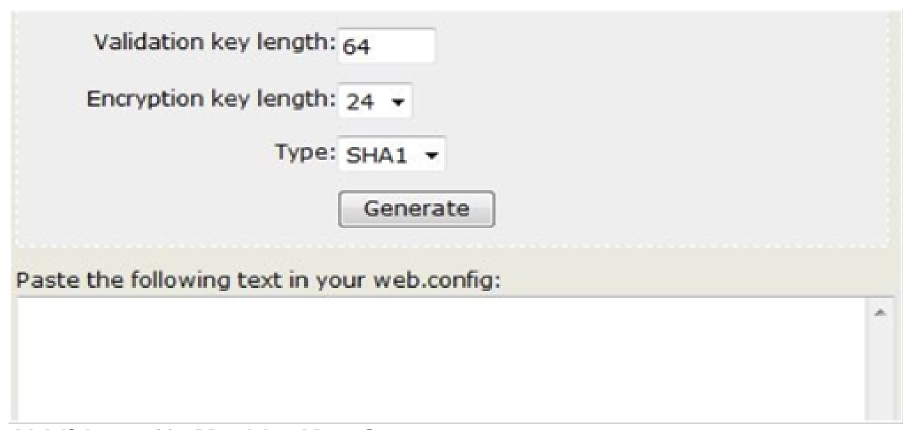


Abbildung 42: MachineKey-Generator

http://www.aspnetresources.com/tools/keycreator.aspx

2.4.52.ASP.NET Page Life-Cycle Diagramm

Leon Andrianarivony hat sich die Mühe gemacht und ein Diagramm des ASP.NET Page Lifce-Cycle angefertig. Das besondere an diesem Diagramm ist, dass sowohl ASP.NET 1.1, als auch ASP.NET 2.0 abgedeckt wird - Ideal als Poster an der Wand.

http://pointerx.net/photos/screenshots/images/852/original.aspx

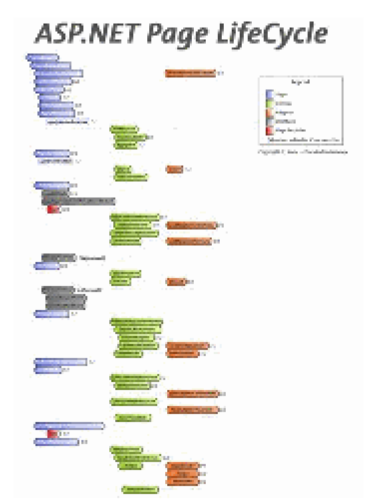


Abbildung 43: ASP.NET LifeCycle

2.4.53.Webseite automatisch aufrufen

Zugegeben, der Titel klingt zunächst etwas komisch, hat aber einen ganz einfachen Hintergrund. In einigen Fällen kann es durchaus hilfreich sein, eine Webseite zu

einem bestimmten Zeitpunkt automatisch aufzurufen, und sei es nur um zu testen ob das Web noch erreichbar ist.

Zunächst benötigt man ein Programm, welches eine Webseite aufrufen bzw. einlesen kann. Hierfür eignet sich hervorragend wget. Durch die umfangreichen Konfigurationsmöglichkeiten kann man viele Gegebenheiten abdecken.

Unter Umständen ist es aber besser ein eigenes Programm zu schreiben und somit sehr flexibel auf neue Anforderungen reagieren zu können. Ein einfaches Konsolenprogramm würde in diesem Fall ausreichen.

Das .NET Framework bietet mit den Klassen `HttpWebRequest` und `HttpWebResponse` eine Möglichkeit an, per Programmcode Webseiten einzulesen und das Ergebnis in einem String zu speichern.

```
static string ReadUrl(string url)
{
    HttpWebRequest webRequest = (HttpWebRequest)WebRequest.Create(url);
    HttpWebResponse webResponse = (HttpWebResponse)webRequest.GetResponse();
    Stream stream = webResponse.GetResponseStream();
    StreamReader streamReader = new StreamReader(stream);
    string html = streamReader.ReadToEnd();

    return html;
}
```

Diese Methode nimmt die URL als Parameter entgegen und stellt per `HttpWebRequest` eine Verbindung her. Das Objekt `webResponse` liest die Rückgabe aus, und speichert diesen per `GetResponseStream` in einem String ab. Existiert die URL nicht, wird eine Exception ausgelöst, die dann abgefangen werden sollte. Denkbar wäre hier eine Benachrichtigung per E-Mail oder ein Eintrag in ein Log-File.

Das Konsolenprogrammm selbst nimmt die geforderte URL als Aufruf-Parameter entgegen.

```
static void Main(string[] args)
{
    if (args.Length == 0)
    {
        Console.WriteLine("Url is missing");
        return;
    }

    Console.WriteLine(ReadUrl(args[0]));
}
```

Um das ausgewählte oder selbst geschriebene Programm zu einem fest definierten Zeitpunkt aufzurufen bietet sich der Task-Manager an. Unter Windows XP findet man diesen unter dem Menüpunkt "Geplante Tasks". Windows Vista benennt diesen Bereich als "Aufgabenplanung". Die Einrichtung hier ist selbsterklärend und kann per Assistenten durchgeführt werden.

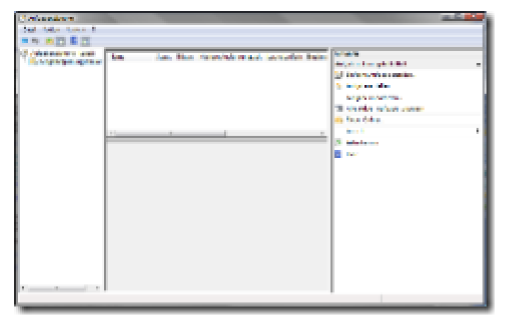


Abbildung 44: Aufgabenplaner

Dieses Beispielprogramm sollte lediglich als Vorlage verwendet und entsprechend der Anforderungen erweitert werden. Wie oben bereits angesprochen, wäre eine Benachrichtigung per E-Mail, ein Datei-Logging, oder das Einlesen der URLs aus einer Textdatei denkbar.

Das Beispielprojekt kann hier heruntergeladen werden.

2.4.54.Einzelne Zellen des GridView einfärben

Mit Hilfe der Einfärbung bestimmter Zellen des GridView in Abhängigkeit zu deren Inhalt, ist es möglich den User direkt auf etwas aufmerksam zu machen.

In diesem Beispiel soll der GridView-Betrachter auf Preise, die jenseits der 200 EUR liegen, hingewiesen werden.

Untitled Page

ID	Name	Price
1	Chair	100,20
2	Table	400,55
3	Desk	150,20
4	Couch	500,60
5	easy chair	100,00

Abbildung 45: Zellen in GridView einfärben

Die Vorgehensweise ist einfach: Nachdem die Zeile eingefügt wurde, muss das Feld `Price` überprüft und entsprechend eingefärbt werden.

Das GridView wirft einen Event mit dem Namen `OnRowDataBound`, nachdem jeweils eine Zeile eingefügt wurde. In diesem Fall ist es also nur erforderlich einen Handler für dieses Event zu definieren und ihn im GridView zu registrieren.

```
<asp:GridView ID="GridView1" runat="server" AutoGenerateColumns="False"
DataKeyNames="ID"
```

```
    DataSourceID="SqlDataSource1" OnRowDataBound="GridView1_RowDataBound">
    <Columns>
        <asp:BoundField DataField="ID" HeaderText="ID" InsertVisible="False"
ReadOnly="True"
            SortExpression="ID" />
        <asp:BoundField DataField="Name" HeaderText="Name"
SortExpression="Name" />
        <asp:BoundField DataField="Price" HeaderText="Price"
SortExpression="Price" />
    </Columns>
</asp:GridView>
```

Dem EventHandler selbst wird ein Objekt vom Typ `GridViewRowEventArgs` übergeben. Dieses Objekt ermöglicht den Zugriff auf die gebundene Zeile.

```
protected void GridView1_RowDataBound(object sender, GridViewRowEventArgs e)
{
    if (e.Row.RowType == DataControlRowType.DataRow)
    {
        DataRowView row = (DataRowView)e.Row.DataItem;

        if (row["Price"] != DBNull.Value)
        {
            if ((decimal)row["Price"] > 200)
            {
                e.Row.Cells[2].BackColor = System.Drawing.Color.Red;
            }
        }
    }
}
```

Zunächst muss überprüft werden, um welchen Typ von gebundener Zeile es sich handelt. Die Einfärbung darf natürlich nur in Daten-Zeilen vorgenommen werden. Anschließend wird das DataItem - welches die Daten der aktuellen Zeile repräsentiert - in ein Objekt vom Typ `DataRowView` gecastet. Über den Feldnamen `Price` erhält man nun Zugriff auf die Daten der Spalte `Price`. Mit Hilfe des Vergleichs `DBNull.Value` muss nun noch überprüft werden, ob das Feld in der Datenbank gefüllt ist. Ist dies der Fall, kann mit dem Vergleich und anschließender Einfärbung begonnen werden.

Das Beispielprojekt kann hier heruntergeladen werden.

2.4.55. ASP.NET Themes per Programmcode ändern

Benutzt eine Web-Anwendung Themes, wird das aktive Theme innerhalb `Web.Config` festgelegt. Über das Property `Page.Theme` ist es außerdem möglich diesen Wert zur Laufzeit zu ändern. Allerdings kann dieses Property nur im `PreInit` Event verändert werden, weiterhin muss die Einstellung bei jedem Request neu gesetzt werden. Ein Http Module schafft in diesem Fall Abhilfe. Einmal in der `Web.Config` registriert, setzt das Modul bei jedem Request das Theme.

Ein Http Module ist ein Assembly, das bei jeder Anforderung einer ASP.NET Seite aufgerufen wird. Weiterhin haben Http Modules die Möglichkeit direkt in die Anforderungspipeline einzugreifen und z.B. aufgrund von bestimmten Zuständen,

definierte Aktionen auszuführen. In diesem Fall wird, wie oben bereits erwähnt, das Theme gesetzt.

Beginnt man mit der Programmierung eines Http Module, muss zunächst das Interface IHttpModule implementiert werden. Visual Studio generiert in diesem Fall die beiden Methoden Init() und Dispose(). An die Methode Init() wird ein Objekt vom Typ HttpApplication übergeben. Mit Hilfe dieses Objekts muss ein neuer EventHandler registriert werden.

```
public void Init(HttpApplication context)
{
    context.PreRequestHandlerExecute += new
EventHandler(context_PreRequestHandlerExecute);
}
```

Der Event PreRequestHandlerExecute wird aufgerufen, bevor ein Http Handler mit der Arbeit beginnt, also auch der Http Handler, der für die Verarbeitung der eigentlichen Seite verantwortlich ist. Im Handler selbst wird versucht eine Instanz der Klasse System.Web.UI.Page auszulesen. Dies schlägt unter Umständen fehl, da ein Http Module nicht nur beim Aufruf einer ASPX Seite aufgerufen wird.

```
void context_PreRequestHandlerExecute(object sender, EventArgs e)
{
    System.Web.UI.Page page = HttpContext.Current.Handler as System.Web.UI.Page;

    if (page != null)
    {
        page.PreInit += new EventHandler(page_PreInit);
    }
}
```

Anschließend kann ein weiterer Handler für den PreInit Event registriert werden. Innerhalb dieses Handlers kann nun das Theme gesetzt werden.

```
void page_PreInit(object sender, EventArgs e)
{
    HttpContext context = HttpContext.Current;

    System.Web.UI.Page page = (System.Web.UI.Page)context.Handler;

    if(context.Session["SelectedTheme"] != null)
        page.Theme = context.Session["SelectedTheme"].ToString(); ;
}
```

Wie man sehen kann, wird der Name des ausgewählten Themes aus dem Session-Objekt ausgelesen. Um Zugriff auf dieses Objekt zu erhalten muss ein weiteres Interface mit dem Namen IReadOnlySessionState implementiert werden. Dieses erlaubt einen lesenden Zugriff auf den Session-Zustand, während das Interface IRequiresSessionState auch den schreibenden Zugriff erlaubt. Mit diesem Schritt ist die Programmierung des Http Module abgeschlossen.

Die Web-Anwendung selbst ist für dieses Beispiel einfach gehalten und besteht lediglich aus einer DropDownListe - mit dem verfügbaren Themes - und einem Button, welches das ausgewählte Theme bestätigt.

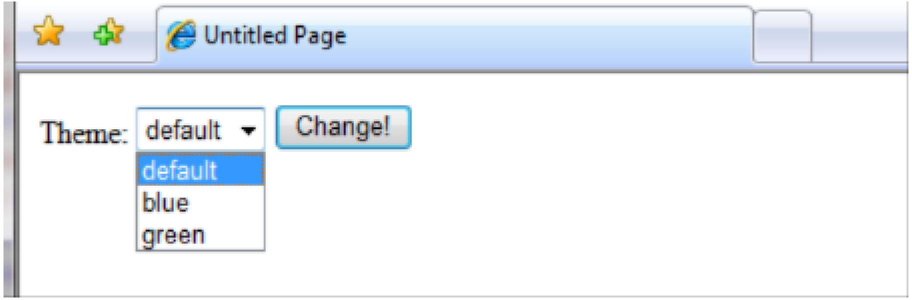


Abbildung 46: Theme Auswahl

Damit das Http Module mit der Arbeit beginnen kann, muss es in der `Web.Config` registriert werden. Dies wird im Abschnitt `HttpModules` erledigt.

```
<httpModules>
  <add name="Tools" type="Tools.ThemeChanger, Tools"/>
</httpModules>
```

Der `OnClick` EventHandler des Buttons speichert das ausgewählte Theme per Session-Objekt und führt anschließend ein Redirect auf die Seite aus. Dies ist nötig, da das Http Module schon längst aufgerufen wurde, bevor der `OnClick` EventHandler das Ereignis bearbeitet hat. Somit kann das Theme erst im zweiten Durchgang ausgelesen werden.

```
protected void cmdChangeTheme_Click(object sender, EventArgs e)
{
    Session["SelectedTheme"] = ddlTheme.SelectedValue;
    Response.Redirect(Request.CurrentExecutionFilePath);
}
```

Anschließend bleibt die Theme-Auswahl des Users erhalten, bis die Session abgelaufen ist. Allerdings ist es ebenfalls sehr einfach dieses Beispiel zu erweitern und die Einstellung in einer Datenbank oder XML Datei zu persistieren.

Das Beispielprojekt kann hier runtergeladen werden.

2.4.56.HyperLink im GridView

Eine häufige Anforderung an eine Anwendung ist die Auflistung von Datenbankinhalten, deren Details auf einer extra Seite dargestellt werden sollen.

Untitled Page

Name	Price
Knife	10,00
Spoon	11,00

Abbildung 47: Hyperlinks in GridView 1

Um diese Anforderungen zu erfüllen sind nur ein paar kleine Schritte notwendig, die per Assistenten erledigt werden können. Selbst programmieren muss man nicht. Zunächst wird eine Datenbank und eine darin vorhandene Tabelle benötigt. In diesem Beispiel ist die Tabelle bewusst sehr einfach gehalten.



Abbildung 48: Hyperlinks in GridView 2

Nachdem eine Verbindung zur Datenbank per `SqlDataSource` Control hergestellt wurde, können die angezeigten Felder des `GridView` über das Kontextmenü und den Punkt `Edit Columns` konfiguriert werden. In diesem Dialog muss ein HyperLinkField Control hinzugefügt werden.

Über die Eigenschaften `DataNavigateUrlFields` und `DataNavigateUrlFormatString` kann nun eingestellt werden welches Datenbankfeld für den Link verwendet werden soll, und wie der Link selbst auszusehen hat. In diesem Fall lautet der Wert für das Feld `DataNavigateUrlField` `Name`, während das Feld `DataNavigateUrlFormatString` mit dem Wert `detail.aspx?id={0}` belegt wird*.* Der Platzhalter `{0}` wird zur Laufzeit mit dem entsprechenden Wert aus der Datenbank ersetzt.

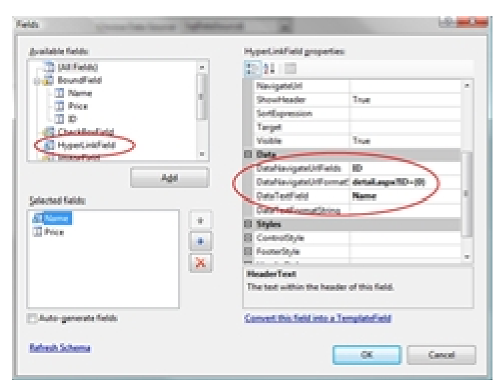


Abbildung 49: Hyperlinks in GridView 3

Das Ergebnis ist die Anzeige des Inhalts als HyperLink. Die Anzeige der Daten auf der Detailseite übernimmt das `DetailsView` Control in Verbindung mit dem `SqlDataSource` Control. Die Url zur Detailseite wurde um den Parameter `id` ergänzt (einem sog. `QueryString`), welcher den zu ladenden Datensatz angibt. Das `SqlDataSource` Control muss so Konfiguriert werden, dass dieser Parameter an die Datenbank weitergereicht und das Ergebnis somit eingeschränkt wird. Im Schritt `Configure the Select Statement` öffnet man hierzu den `WHERE` *Dialog* und legt einen entsprechenden Übergabeparameter an.

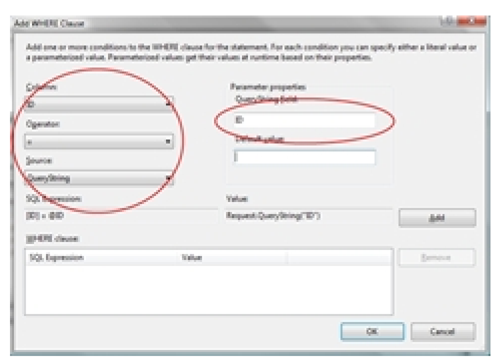


Abbildung 50: Hyperlinks in GridView 4

Wird diese Seite mit dem ID-Parameter aufgerufen erhält man als Ergebnis den Inhalt des Datensatzes. Ohne diesen Parameter bleibt die Anzeige leer. In diesem Fall empfiehlt es sich zusätzlich eine Abfrage zu implementieren und eine Meldung auszugeben oder die Anforderung per `Response.Redirect` umzuleiten.

Den Download des Beispielprojekts findet man hier.

2.4.57. ASP.NET: Publish einer Website aus der Konsole (command line)

Im Zuge eines automatisierten Deploymentprozesses möchte man eventuell eine vorkompilierte Version der ASP.NET Anwendung in ein bestimmtes Verzeichnis generieren lassen. Hierfür kann das Tool **aspnet_compiler.exe**, welches seit dem .NET Framework 2.0 verfügbar ist, verwendet werden.

Der Aufruf sieht folgendermaßen aus:

```
aspnet_compiler -v /MyIISWebApplication MyTargetPath
```

MyIISWebApplication muss als solche im IIS registriert sein. **MyTargetPath** gibt den Pfad an, in welchen die vorkompilierte Anwendung gespeichert werden soll. Hier sind vor und nach der Pfadangabe Anführungszeichen zu setzen, wenn der Pfad Leerzeichen enthält.

Gegebenenfalls muss der Pfad zu `aspnet_compiler.exe` in die Umgebungsvariablen hinzugefügt werden, damit das Tool global aufgerufen werden kann. Zu finden ist das Tool unter:

```
C:\{windir}\Microsoft.NET\Framework\v2.0.50727
```

Aus dem Zielverzeichnis kann nach der Generierung die vorkompilierte Version entnommen und verwertet werden.

Weitere Commandline-Schalter und zusätzliche Informationen können im MSDN gefunden werden.

2.4.58.ASP.NET: URL aus lokalem Pfad generieren

Mit `Server.MapPath` kann sehr einfach der lokale Pfad zu einer Datei generiert werden. In manchen Fällen ist jedoch der umgekehrte Weg notwendig: Aus einem vorhanden Pfad soll eine URL erstellt werden. Nachfolgendes Code-Snippet zeigt, wie dies geht:

```
private String ReverseMapPath(String PhysicalFilePath)
{
    String AppPath = HttpContext.Current.Server.MapPath("~");
    String url = String.Format
        ("~{0}",
         PhysicalFilePath.Replace(AppPath, "").Replace("", "/"));
    return url;
}
```

2.4.59.ASP.NET: Session-Informationen zur richtigen Zeit auslesen

Für eine Webanwendung werden meist Sessions benötigt. Darin werden diverse Informationen abgelegt, um von unterschiedlichen Seiten darauf zugreifen zu können. Bei der Verwendung von eigenen UserControls kann es - vor allem für ASP.NET Neulinge - sehr schnell zu einigen Missverständnissen und Fehlverhalten kommen. Daher möchte ich anhand eines kurzen Beispiels erklären, worauf generell zu achten ist.

Ausgangssituation

Erstellt werden soll eine kleine Webanwendung, welche aus einem simplen Menü und einem Login besteht. Sowohl das Menü, als auch das Login wird über ein von *System.Web.UI.UserControl* abgeleitetes Steuerelement dargestellt. Nach einem erfolgreichen Login, soll das Menü weitere, zusätzliche, Menüpunkte aufweisen.

Umsetzung

Da für diese Anwendung eine Session benötigt werden, wurde eine eigene Klasse *SessionData* erstellt, die grundlegende und für die Anwendung wichtige Informationen enthält. Hauptsächlich handelt es sich dabei um Informationen zum User, die an unterschiedlichen Stellen benötigt werden und nicht überall nachgeladen werden sollen.

Zusätzlich wurde eine Klasse *SessionList* erstellt, welche alle aktuellen Sessions zugeteilt bekommt und bestimmte Funktionalitäten aufweist, die für das Session-Handling notwendig sind (Prüfung auf Gültigkeit, ist die Session bereits abgelaufen, usw.). Diese Klasse wird in der *Global.asax* im Eventhandler *Application_Start* instanziiert.

Die *Default.aspx* stellt die Haupteinstiegs-Seite dar und enthält - wie bereits oben erwähnt - dein Menü-Control und ein Login-Control, die beide selbst erstellt wurden und daher auch ein bestimmtes notwendiges Verhalten an den Tag legen.

Sehen wir uns nun den Sourcecode der *Default.aspx* genauer an:

```
private SessionData _sessionData = null;
protected void Page_PreRender(object sender, EventArgs e)
{
    _sessionData = Session["sessionInfo"] as SessionData;
    if (Session["sessionInfo"] != null)
    {
        _sessionData = (SessionData)Session["sessionInfo"];
        foreach (Control c in ContentPlaceHolder.Controls)
        {
            if (c.ID == "loginControl")
            {
                ContentPlaceHolder.Controls.Remove(c);
                break;
            }
        }
    }
}
protected void Page_Init(object sender, EventArgs e)
{
    if (_sessionData == null)
    {
        Control loginControl = LoadControl("modules/LoginControl.ascx");
        loginControl.ID = "loginControl";
        this.ContentPlaceHolder.Controls.Clear();
        this.ContentPlaceHolder.Controls.Add(loginControl);
    }
    else
    {
        this.ContentPlaceHolder.Controls.Clear();
    }
    Control menuControl = LoadControl("modules/MenuControl.ascx");
    menuControl.ID = "menuControl";
    MenuPlaceHolder.Controls.Clear();
    MenuPlaceHolder.Controls.Add(menuControl);
}
protected void Page_Load(object sender, EventArgs e)
{

}
```

Wie zu sehen ist, werden grundsätzlich zwei Events bedient (Page_Load wurde nur angeführt um zu zeigen, dass sich hierin keinerlei Sourcecode befindet):

- Page_Init
- Page_PreRender

Wie im Artikel ASP.NET: Die PostBack-Falle unter **Lebensyklen** beschrieben, müssen wir bei unserem Source nun darauf aufpassen, welches Event wann gefeuert und welche Methode wann aufgerufen wird. Kommen UserControls ins Spiel, muss natürlich deren Lebenszyklus ebenfalls betrachtet werden (UserControls unterliegen den gleichen Lebenszyklen wie Seiten).

Im initialen Aufruf der *Default.aspx* wird zuerst *Page_Init* aufgerufen. Darin wird festgestellt, dass es keine aktuelle Session gibt und das *LoginControl* geladen. Dieses enthält folgenden Sourcecode:

```
public partial class LoginControl
    : System.Web.UI.UserControl
{
    protected void Page_Load(object sender, EventArgs e)
    {
        LoginButton.Click += new EventHandler(LoginButton_Click);
    }
    protected void LoginButton_Click(object sender, EventArgs e)
    {
        // Implementierung Login
    }
}
```

Sobald also das LoginControl instanziiert wird, werden die einzelnen Events durchlaufen. In diesem Fall *Page_Load*. An dieser Stelle wird lediglich der Button-Handler gesetzt. Es passiert noch nichts großartiges.

Wenn wir wieder zurück zur Default-Seite gehen, wird nach dem Laden des Login-Controls das Menü geladen. Dieses enthält lediglich ein originales ASP.NET MenuControl und folgenden Source:

```
protected void Page_PreRender(object sender, EventArgs e)
{
    if (Session["sessionInfo"] == null)
    {
        foreach (MenuItem mi in MainMenu.Items)
        {
            if (mi.Value != "Home")
            {
                mi.Enabled = false;
                mi.Selectable = false;
            }
        }
    }
    else
    {
        foreach (MenuItem mi in MainMenu.Items)
        {
            mi.Enabled = true;
            mi.Selectable = true;
        }
    }
}
protected void Page_Load(object sender, EventArgs e)
{

}
```

Hier wird nun im Page_PreRender überprüft, ob bereits eine Session gesetzt wurde. Dies ist vor dem Einloggen nun auch nicht Fall, wodurch bis auf den Home-Eintrag alle weiteren deaktiviert werden.

Nun kommt es zum Login. Die Daten werden per Post an den Server übertragen. Hier wird nun folgende Reihenfolge ausgeführt:

1. Default.aspx: Page_Init
2. LoginControl.ascx: Page_Load
3. Default.aspx: Page_PreRender
4. MenuControl.ascx: Page_PreRender

Verfolgt man diese Reihenfolge im Sourcecode, ist schön zu sehen, wie und vor allem wann Session-Informationen ausgelesen werden müsssen, um korrekt angewandt zu werden. Daher verweise ich nochmals an den Artikel ASP.NET: Die PostBack-Falle, welcher den Lebenszyklus von Seiten genau auflistet.

Fazit

Dieser Artikel sollte sein, wie mit Session-Informationen unter Verwendung von UserControls umgegangen und worauf geachtet werden soll. Unter Berücksichtigung des Lebenszykluses der Seite bzw. der gesamten Webanwendung (als auch der verwendeten UserControls) lassen sich dadurch entstehende Probleme schnell eingrenzen und lösen.

2.4.60.ASP.NET: Die PostBack-Falle

Irgendwann tappt jeder in die PostBack-Falle. So ist es auch mir passiert und daher dieser kurze Eintrag zu diesem Thema. Der Beginn macht eine kurze Einführung.

Was ist ein PostBack

Unter einem PostBack versteht man das senden einer Seite zum Server unter Verwendung des HTTP-Vers **Post**. Die Eingaben der gesendeten Seite werden vom Server verarbeitet und eine neue Seite generiert. Die gesamte Seite wird neu geladen.

Konkretes Problem

Beim Laden einer Seite (vor allem wenn Werte zugewiesen werden) sollte daher abgefragt werden, ob es sich tatsächlich um ein PostBack handelt. Hier ein konkreter Fall:

```
protected void Page_Load(object sender, EventArgs e)
{
    FirstnameTextBox.Text = "Norbert";
    LastnameTextBox.Text = "Eder";
}
```

In diesem Fall wird ein PostBack nicht abgefragt. Nun werden beim Laden der Seite die vorhandenen Textfelder mit Werten befüllt. Werden diese verändert und wieder an den Server übertragen, werden die Änderungen beim neu Erstellen der Seite wieder überschrieben. Die geänderten Werte scheinen also am Client nicht mehr auf.

Grund hierfür ist, dass die Daten in der Methode *LoadPostData* (siehe Lebenszyklen etwas weiter unten) geladen und im anschließenden *Load*-Event überschrieben werden. Anders sieht dies aus, wenn die Wertezuweisung durch die Abfrage bezgl. eines PostBacks gesichert wird:

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!this.IsPostBack)
    {
        FirstnameTextBox.Text = "Norbert";
        LastnameTextBox.Text = "Eder";
    }
}
```

In diesem Fall werden die Daten korrekt am Client angezeigt - nämlich die geänderten.

Zu guter Letzt möchte ich noch die unterschiedlichen Lebenszyklen einer Seite anführen, da sich diese durch ein PostBack verändern und darauf entsprechend Rücksicht genommen werden muss.

Lebenszyklen

Erster Aufruf
Die nachfolgenden Ereignisse bzw. Methoden werden beim initialen Aufruf einer Seite der Reihe nach abgearbeitet:

1. Init
2. Load
3. PreRender
4. SaveViewState
5. Render
6. Dispose
7. Unload

Bei einem PostBack

Bei einem PostBack erweitert sich der Lebenszyklus einer Seite geringfügig. Hier nun die neue Reihenfolge:

1. Init
2. LoadViewState
3. LoadPostData
4. Load
5. RaisePostDataChangedEvent
6. RaisePostBackEvent
7. PreRender
8. SaveViewState
9. Render
10. Dispose
11. UnLoad

Fazit

Es gilt sehr gut zu überlegen, in welchen Events welche Funktionalität ausgeführt werden soll. Hierfür, solange man sich noch nicht daran gewöhnt hat, regelmäßig einen Blick auf den Lebenszyklus werfen.

2.4.61.ASP.NET: Web Controls dynamisch laden

In vielen Situationen ist es notwendig, Web Controls dynamisch in einen Platzhalter (Placeholder-Control) zu laden. Dies kann sehr einfach bewerkstelligt werden:

```
this.ContentPlaceHolder.Controls.Clear();
this.ContentPlaceHolder.Controls.Add
    (
    LoadControl("modules/TestControl.ascx")
    );
```

Das Beispiel geht davon aus, dass Web Controls im Verzeichnis **`modules`** gespeichert sind. Beim `ContentPlaceHolder` handelt es sich um ein `ContentHolder`-Control.

Natürlich kann dies auch entsprechend dynamischer gestaltet werden, da sich das zu ladende Control eventuell aus dem aufgerufenen Menüeintrag oder anderen Informationen definieren lässt.

2.4.62.Verzeichnisse per ASP.NET FormsAuthentication in 60 Sekunden schützen

Inspiriert durch einen Thread auf www.aspnetzone.de soll dieser Beitrag zeigen, wie schnell sich eine Web-Applikation oder nur ein einfaches Verzeichnis per FormsAuthentication schützen lässt.

Zunächst ist es nötig die *Web.Config* anzupassen. In diesem Beispiel soll das Verzeichnis *Secure* geschützt werden. Weiter wird eingestellt das Session-Cookies verwendet werden dürfen und diese noch für 120 Minuten nach der letzten Aktion gültig sein sollen.

```
<authentication mode="Forms">
 <forms cookieless="UseCookies" defaultUrl="~/Secure/SecureSite.aspx"
loginUrl="~/default.aspx" timeout="120">
  </forms>
</authentication>
```

Nun müssen für das Verzeichnis die Berechtigungen gesetzt werden. Dies erfolgt über den *Location* Node unter Angabe des Verzeichnisnamens. Mit Hilfe der Angabe *deny* wird angegeben das nur bekannte - also autorisierten User – Zugriff erhalten sollen.

```
<location path="Secure">
  <system.web>
```

```
    <authorization>
      <deny users="?"/>
    </authorization>
  </system.web>
</location>
```

Nachdem die Zugangsdaten durch eine selbstgeschriebene Routine validiert wurden, kann der Login ausgeführt werden. Hierfür bietet sich die Methode *FormsAuthentication.RedirectFromLoginPage()* an, die nach dem setzen des Session-Cookies eine Redirect auf die angeforderte Seite des Users durchführt.

```
protected void btt_Click(object sender,
  EventArgs e)
{
   FormsAuthentication.RedirectFromLoginPage("TestUser", false);
}
```

Nachdem diese Schritte durchgeführt wurden, ist die Einrichtung bereits abgeschlossen und das Web oder Verzeichnis gesichert.

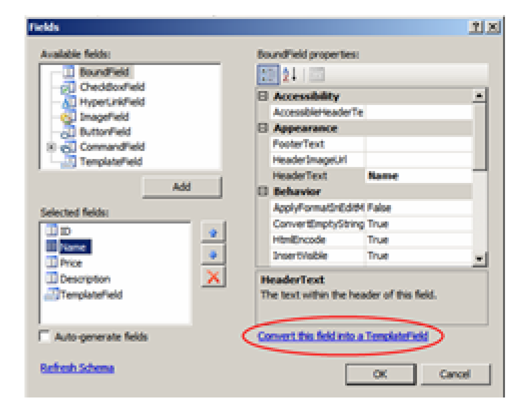
Jürgen hat in seinem Weblog eine weitere Möglichkeit aufgezeigt, wie sich ein Web vor fremden Zugriffen schützen lässt.

Das Beispielprojekt zu diesem Beitrag findet sich hier.

2.4.63.Eingaben im DetailsView validieren

Ein häufig verwendetes Control um Daten anzuzeigen, einzufügen, oder zu editieren ist das DetailsView Control. Insbesondere vom User eingegebe Daten sollten überprüft werden, bevor sie in die Anwendung übertragen werden. Allerdings beinhalten die definierten Felder im DetailsView keine Eigenschaft, die eine Validierung aktiviert. Auch die mitgelieferten ASP.NET Validation Controls können ohne einen kleinen Trick nicht verwendet werden.

Der bereits angesprochene Trick ist in diesem Fall der kleine Link "*Convert this into a TemplateField*" im *Fields* Menü des DetailsView Control.



Wird dieser geklickt, ist das Ergebnis eine einfache TextBox, eingeschlossen in einem TemplateField. Innerhalb dieses Feldes können nun die Validation Controls benutzt werden. Natürlich ist es auch möglich die Controls außerhalb des Template Field zu plazieren.

```
<asp:TemplateField HeaderText="Name" SortExpression="Name">
 <EditItemTemplate>
  <asp:RequiredFieldValidator ID="valTextBox1" runat="server"


   ControlToValidate="TextBox1" ErrorMessage="Error1" />
  <asp:TextBox ID="TextBox1" runat="server" Text='<%#
   Bind("Name") %>'></asp:TextBox>
 </EditItemTemplate>
 <InsertItemTemplate>
  <asp:RequiredFieldValidator ID="valTextBox1" runat="server"


   ControlToValidate="TextBox1" ErrorMessage="Error1" />
  <asp:TextBox ID="TextBox1" runat="server" Text='<%#
   Bind("Name") %>'></asp:TextBox>
 </InsertItemTemplate>
 <ItemTemplate>
  <asp:Label ID="Label1" runat="server" Text='<%#
   Bind("Name") %>'></asp:Label>
 </ItemTemplate>
</asp:TemplateField>
```

Auf diese Weise können wie gewohnt die Eingaben auch im DetailsView validiert werden. Bei genauerer Betrachtung wäre sogar eine Unterscheidung zwischen Einfügen u. Editieren möglich.

Das Beispielprojekt zu diesem Beitrag kann hier runtergeladen werden.

2.4.64.UserControls um eigene Eigenschaften erweitern

UserControls bieten eine hervorragende Möglichkeit um wiederholende Funktionalität oder Quellcode auszulagern. Häufig ändert sich aber dennoch ein kleiner Teil von Seite zu Seite. Um diese Anforderung abbilden zu können und die Flexibilität das Controls zu erhöhen, ist es möglich UserControls um eigene Eigenschaften zu erweitern.

In diesem einfachen Beispiel soll das UserControl einen Text über die Eigenschaft *MyValue* engegennehmen und diesen über ein Literal Control ausgeben.

```
<UC:WebUserControl ID="uc1" runat="server" MyValue="Test
   1" />
```

Hierfür ist nichts weiter nötig als in der Code-Behind Datei des UserControls ein neues Property hinzuzufügen.

```
private string myValue;
public string MyValue
{
 get { return myValue;
   }
 set {
   myValue = value; }
```

```
}
```

Die eigentliche Funktionalität um den Text auszugeben ist nun nur noch Formsache.

```
protected void Page_Load(object sender, EventArgs e)
{
  lit.Text = this.myValue;
}
```

Wird dieses UserControl nun im Kopf einer Webseite oder [per Web.Config registriert](), steht die Eigenschaft - welche im Code-Behind angelegt wurde - sofort im IntelliSense zur Verfügung.

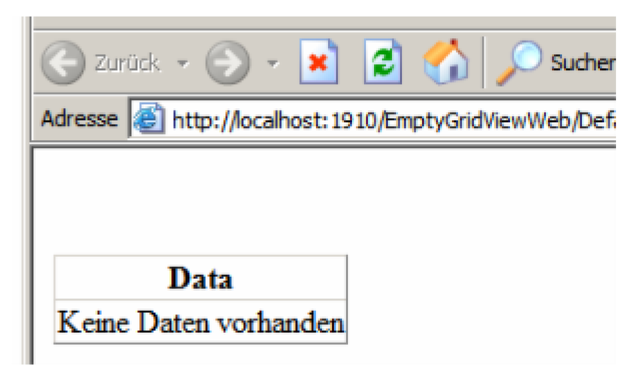
```
<UC:WebUserControl ID="uc1" runat="server" m />
<br />
<br />
<UC:WebUserControl ID="uc2" runat="serve
iiv>
:orm>
```

EnableTheming
EnableViewState
ID
MyValue
OnAbortTransaction

Den Download des Beispielprojekts findet man [hier]().

2.4.65.GridView anzeigen obwohl keine Daten vorhanden sind

Das GridView Control eignet sich hervorragend um Daten jeglicher Art anzuzeigen. Sind allerdings keine Daten vorhanden, wird das gesamte Control ausgeblendet. Eine häufige Anforderung ist es jedoch einen Text, welcher auf die nicht vorhandenen Daten hinweist und die Überschriften, der Tabelle anzuzeigen.



Um dieses Problem zu lösen bietet sich ein CustomControl an, welches vom GridView abgeleitet und um die benötigte Funktionalität erweitert wird. Die benötigten Funktionen definieren sich wie folgt:

- Möglichkeit das GridView trotz leerer Daten anzuzeigen
- Eingabe eines Textes, der bei leeren Daten angezeigt wird

Zunächst müssen die zusätzlichen Eigenschaften definiert werden. Die Eingaben sollten im ViewState abgelegt werden, damit diese auch nach einem PostBack zur Verfügung stehen.

```
public bool ShowEmptyTable
{
```

```
    get
    {
        object o = ViewState["ShowEmptyTable"];
        return (o != null ? (bool)o : true);
    }
    set
    {
        ViewState["ShowEmptyTable"] = value;
    }
}

public string EmptyText
{
    get
    {
        object o = ViewState["EmptyText"];
        return (o != null ? o.ToString() : "");
    }
    set
    {
        ViewState["EmptyText"] = value;
    }
}
```

Anschließend wird die Methode *CreateChildControls* überschrieben und angepasst. Sind keine Zeilen an das GridView gebunden und die Eigenschaft *ShowEmptyTable* gesetzt, wird zuerst der Kopf der Tabelle inkl. der Überschriften eingefügt. Weiter wird eine leere Zeile hinzugefügt, die den Text der Eigenschaft *EmptyText* anzeigt.

```
protected override int CreateChildControls(System.Collections.IEnumerable
dataSource, bool dataBinding)
{
    int numRows = base.CreateChildControls(dataSource, dataBinding);

    if (numRows == 0 && ShowEmptyTable)
    {
        Table table = new Table();
        table.ID = this.ID;

        GridViewRow row = base.CreateRow(-1, -1, DataControlRowType.Header,
DataControlRowState.Normal);

        DataControlField[] fields = new DataControlField[this.Columns.Count];
        this.Columns.CopyTo(fields, 0);
        this.InitializeRow(row, fields);
        table.Rows.Add(row);

        row = new GridViewRow(-1, -1, DataControlRowType.DataRow,
DataControlRowState.Normal);
        TableCell cell = new TableCell();
        cell.ColumnSpan = this.Columns.Count;
        cell.Width = Unit.Percentage(100);
        cell.Controls.Add(new LiteralControl(EmptyText));
        row.Cells.Add(cell);
        table.Rows.Add(row);

        this.Controls.Add(table);
    }

    return numRows;
}
```

Um das abgeänderte GridView zu verwenden, muss es um Webprojekt referenziert und per Web.Config registriert werden.

```
<controls>
  <add assembly="EmptyGridView" namespace="EmptyGridView" tagPrefix="CC"/>
</controls>
```

Ist dies geschehen, können die Eigenschaften direkt per IntelliSense aufgerufen und gesetzt werden. In diesem Beispiel wird eine leere Tabelle mit dem Namen *MyData* an das GridView gebunden. Die Tabelle enthält nur eine Spalte mit dem Namen *Data*.

```
<CC:EmptyGridView id="grid" runat="server" ShowEmptyTable="True" EmptyText="Keine
Daten vorhanden" AutoGenerateColumns="False" DataSourceID="SqlDataSource1" >
    <Columns>
        <asp:BoundField DataField="Data" HeaderText="Data"
SortExpression="Data" />
    </Columns>
</CC:EmptyGridView>
```

Das abgeänderte GridView Control kann in mehreren Projekten eingesetzt werden und bietet den vollen Funktionsumfang des eigentlichen GridView Controls. Das gesamte Projekt inkl. Beispielapplikation kann hier runtergeladen werden. Um das EmptyGridView Control direkt einsetzten, reicht dieser Download.

2.5.Services

2.5.1.Windows Dienste mit C# und .NET 2.0 kontrollieren

Hier eine kleine Demoklasse die den Umgang mit Windows-Diensten zeigt. Die Klasse selbst bietet nur die Möglichkeit den Status eines Dienstes abzufragen und diesen zu Starten bzw. zu Stoppen. Weitere Möglichkeiten kann den Klassen `ServiceController` entnommen werden.

```
public class ProcessHandler
{
    private string processName = null;

    public string ProcessName
    {
        get { return this.processName; }
        set { this.processName = value; }
    }

    public ServiceControllerStatus GetProcessState()
    {
        ServiceController sc = new ServiceController(processName);
        if (sc != null)
        {
            return sc.Status;
        }
        return ServiceControllerStatus.Stopped;
    }

    public void StartProcess()
```

```
    {
        ServiceController sc = new ServiceController(processName);
        if (sc != null)
            sc.Start();
    }

    public void StopProcess()
    {
        ServiceController sc = new ServiceController(processName);
        if (sc != null && sc.CanStop)
            sc.Stop();
    }

}
```

2.5.2.Webservice-Methoden überladen

In modernen Programmiersprachen können Methoden überladen werden. Somit ist es möglich verschiedene Funktionalitäten abzubilden, ohne einen weiteren Methoden-Namen zu erstellen.

```
public string HelloWorld()
{
 return "Hello";
}
public string HelloWorld(string yourName)
{
 return "Hello
   " + yourName;
}
```

Möchte man diese Methoden allerdings für einen Webservice zur Verfügung stellen, endet dies mit einer unschönen Fehlermeldung.

Both X and Y use the message name 'Z'. Use the MessageName property of the WebMethod custom attribute to specify unique message names for the methods.

Der Grund hierfür ist, dass Methodennamen innerhalb eines Webservice immer eindeutig sein müssen.
Dieses Problem kann mit dem Attribut `MessageName` gelöst werden.

```
[WebMethod (MessageName="HelloWorld")]
public string HelloWorld()
{
 return "Hello";
}
[WebMethod (MessageName = "HelloWorldWithName")]
public string HelloWorld(string yourName)
{
 return "Hello
   " + yourName;
}
```

Nun muss der Umstand, einen weiteren Methodennamen zu kreieren, allerdings wieder in Kauf genommen werden.

2.6.Windows Presentation Foundation

2.6.1.Windows Presentation Foundation - Teil 2: XAML und Layouts

Im zweiten Teil der Tutorials-Reihe über die Windows Presentation Foundation werden die Themen XAML und Layouts behandelt. So wird beschrieben, was XAML genau ist und wofür es da ist. Weiters wird gezeigt, welche grundlegenden Layout-Elemente zur Verfügung stehen. Natürlich ist auch dieser Teil wieder voll von Beispiel-Code und Screenshots zur Veranschaulichung.

Windows Presentation Foundation - Teil 2: XAML und Layouts (PDF)

2.6.2.WPF: Rotation und Skalierung einfach gemacht

Beschäftigt man sich näher mit der Materie Windows Presentation Foundation, sieht man schon an sehr einfachen Beispielen, dass die Möglichkeiten schon sehr mächtig sind. Beispielhaft zeige ich an dieser Stelle, wie einfach ein Button skaliert und gedreht werden kann.

Dazu wird im Beispiel ein simpler Button erstellt und plaziert. Über Schieberegeler ist es möglich, die Values für die Rotation bzw. des Zoomfaktors zu setzen. Dies sieht dann folgendermaßen aus:

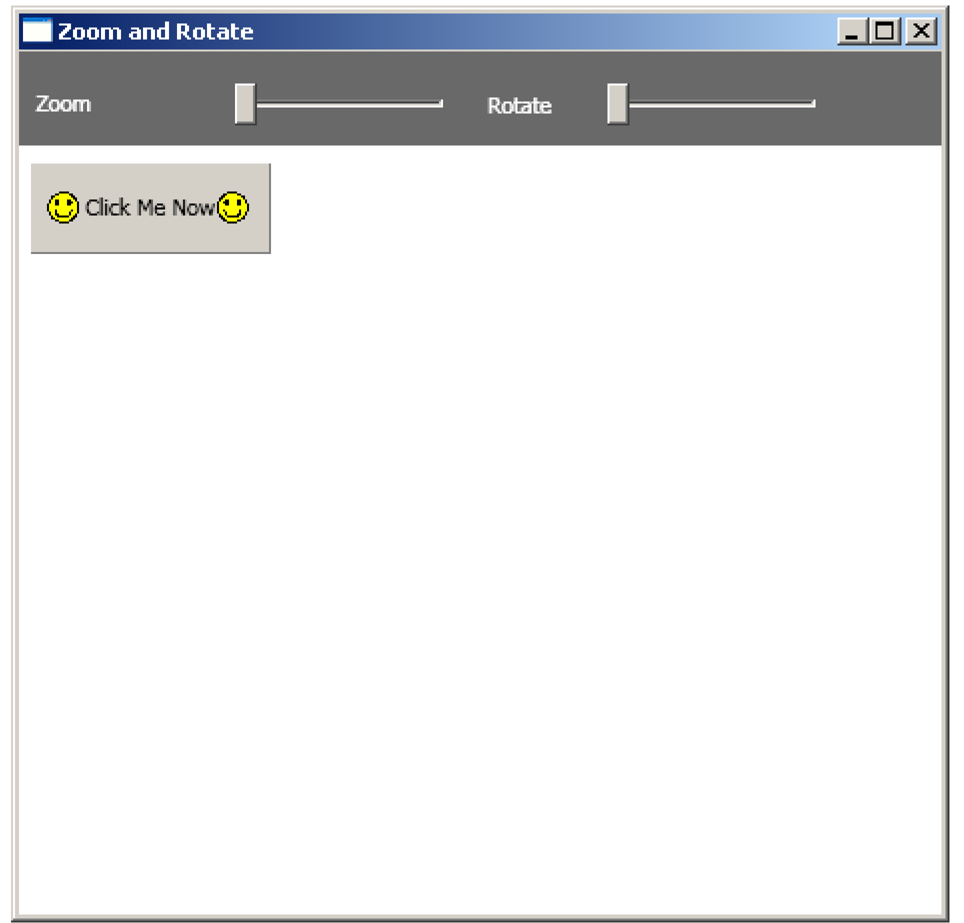


Abbildung 51: WPF Rotation 1

Um nun die Funktionalität zu implementieren ist nichts weiter zu machen, als die entsprechenden EventHandler zu setzen:

Zoomfaktor

```
public void sliderZoom_ValueChanged(object sender, RoutedEventArgs e)
{
    ScaleTransform st = new ScaleTransform(sliderZoom.Value, sliderZoom.Value);
    this.btnTest.LayoutTransform = st;
}
```

Rotationsfaktor

```
public void sliderRotate_ValueChanged(object sender, RoutedEventArgs e)
{
    RotateTransform rt = new RotateTransform(sliderRotate.Value);
    this.btnTest.RenderTransform = rt;
}
```

Das Ergebnis sieht dann wie folgt aus:



Abbildung 52: WPF Rotation 2

Dies kann jetzt nicht nur auf einen Button angewandt werden, sondern auf die gesamte Oberfläche, oder auch nur einzelne Bereiche. Der dahinter liegende XAML-Code etc. ist im gesamten Projekt enthalten, welches zum Download bereit steht.

Download WPF Rotation Test Beispiel

2.6.3.Windows Presentation Foundation - Teil 1: Einführung

Der erste Teil einer Tutorials-Reihe gibt eine kleine Einführung in das Thema Windows Presentation Foundation (ehemals Avalon) und beschreibt die wichtigsten Punkte. Zudem wird anhand einer sehr simplen Testanwendung gezeigt, wie eine solche entwickelt werden kann.

Der nächste Teil dieser Serie wird weiter in die Tiefe gehen und Hintergründe näher erläutern.

Windows Presentation Foundation - Teil 1: Einführung (PDF)

2.6.4.Windows Workflow Foundation Web Workflow Approvals Starter Kit

Microsoft hat ein Starter Kit veröffentlicht, welches die Verwendung der Windows Workflow Foundation in Web-Anwendungen zeigen soll.

This starter kit is a Visual Studio 2005 project that demonstrates using Windows Workflow Foundation for simple task oriented workflow in an ASP.NET web application. A workflow model is used to automate work order requests at a small example company. It includes three pre-defined roles which each play a part in the work order creation, approval and monitoring. The starter kit may be modified for other workflow models to suit other small web based task management systems.

Den Download und weitere Infos gibt es hier.

Mehr Informationen zur Windows Workflow Foundation gibt es unter folgenden Links:
Windows Workflow Foundation (WF) Tutorials And Samples
Hands-on Labs for Windows Workflow Foundation
Die Microsoft Windows Workflow Foundation - Eine Einführung für Entwickler

2.6.5.WPF in Windows Forms verwenden

In einem kleinen Projekt musste ich WPF-Controls in einer Windows Form verwenden. Wer vor eben solcher Geschichte steht, sollte einen Blick auf den Artikel Hosting a WPF Control in a Windows Forms Application. Interessante Sache.

Edit: Da dieser Artikel noch auf eine Beta aufbaut, möchte ich hier kurz die notwendigen Schritte erläutern, da sich doch etwas geändert hat.

WPF-Control erstellen

Im ersten Schritt muss ein entsprechendes WPF-Control erstellt oder ein vorhandenes benutzt werden. In meinem Beispiel habe ich ein einfaches UserControl erstellt.

```
<UserControl x:Class="AddinTestApp.LoginControl"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Height="118" Width="300">
    <Grid Height="120">
       <Label Height="23" HorizontalAlignment="Left"
            Margin="9,19,0,0" Name="label1"
            VerticalAlignment="Top" Width="120">
            Username
       </Label>
       <Label Height="23" HorizontalAlignment="Left"
            Margin="9,40,0,0" Name="label2"
            VerticalAlignment="Top" Width="120">
            Password
       </Label>
       <TextBox Height="21" Margin="91,21,20,0"
            Name="UsernameTextBox"
            VerticalAlignment="Top" />
       <TextBox Height="21" Margin="91,44,20,0"
            Name="PasswordTextBox"
            VerticalAlignment="Top" />
       <Button Height="23" HorizontalAlignment="Right"
            Margin="0,77,20,0" Name="LoginButton"
            VerticalAlignment="Top" Width="75">
            Login
```

```
        </Button>
    </Grid>
</UserControl>
```

Windows Formular erstellen

Im zweiten Schritt wird ein normales Windows-Forms-Formular erstellt. Im selben Projekt müssen nun einige Referenzen hinzugefügt werden:

- PresentationCore
- PresentationFramework
- UIAutomationProvider
- UIAutomationTypes
- WindowsBase

Zusätzlich ist noch die Assembly WindowsFormsIntegration.dll zu laden. Diese befindet sich für gewöhnlich im Ordner:

```
%programfiles%\Reference Assemblies\Microsoft\

Framework\v3.0\WindowsFormsIntegration.dll
```

WPF-Control zur Anzeige bringen

Wenn nun sowohl das WPF-Control als auch das Windows Formular erstellt wurden, kann das WPF-Control folgendermaßen eingebunden werden:

```
private void MainForm_Load(object sender, EventArgs e)
{
     ElementHost host = new ElementHost();
     LoginControl lc = new LoginControl();
     host.Child = lc;

     host.Dock = DockStyle.Fill;
     this.Controls.Add(host);
}
```

Bei einem Start der Anwendung sollte nun das WPF-Control sichtbar sein. Wichtig hierbei ist, dass das `Host`-Element via `DockStyle.Fill` am Windows Formular ausgerichtet werden muss.

2.6.6.WPF Serie Teil 1: Die Windows Presentation Foundation aus der Sicht eines Unternehmens

Wie bereits angekündigt, werde ich hier eine Umsetzungs-Serie zum Thema WPF starten. Im Zuge dieser Serie werden einige interessante Themen besprochen.

Eine Einführung zum Thema Windows Presentation Foundation werde ich an dieser Stelle nicht bieten, hierfür müssen die beiden von mir erstellen Tutorials zu diesem Thema reichen:

[Windows Presentation Foundation - Teil 1: Einführung](#)

[Windows Presentation Foundation - Teil 2: XAML und Layouts](#)

Vielmehr möchte ich hinterfragen, welchen Stellenwert WPF in Unternehmen hat und welche Überlegungen mitspielen, um auf WPF umzusteigen.

Auch heute noch - WPF gibt es nun ja schon länger - wird die Windows Presentation Foundation als neue Technologie gehandelt. Beispiele gibt es dazu ja bereits einige und durch Silverlight hat XAML sicherlich einen neuen Hype erfahren. Dennoch scheuen sich sehr viele Unternehmen diese Technologie einzusetzen. Warum ist dem so?

Hier spielen sicherlich mehrere Faktoren eine wichtige Rolle:

Nicht jedes Unternehmen ist dazu gemacht, ein Innovator bzw. ein früher Adopter zu sein (siehe Erklärung). Dies bedeutet, nicht jede Firma setzt auf neueste Technologien. Aus unterschiedlichsten Gründen: Viele vertrauen auf Bewährtes. Es bestehen wenig Risiken (damit haben sich bereits Jahre zuvor andere auseinander gesetzt), Informationen sind breit verfügbar (Foren, Blogs, Bücher) und es gibt durchaus genügend Entwickler die sich mit Bestehendem auskennen und somit im Notfall eingesetzt werden können. Bei einem Innovator sieht es hingegen anders aus: Aktuellste Technologien werden eingesetzt um der Konkurrenz gegenüber einen technologischen Vorteil zu schaffen. Informationen sind rar (SDK Dokumentation, wenn verfügbar). Know-How-Träger müssen kostenintensiv aufgebaut werden, wodurch die Produktivität anfangs sinkt und natürlich das Risiko besteht, das vorgesehene Projekt nie abzuschließen.

Eng mit dem ersten Punkt ist die Tatsache, dass Zeit geschaffen werden muss, um sich eine Technologie anzueignen. In Zeiten wie diesen - Microsoft veröffentlicht laufend neue Technologien - ist es sehr schwierig mit den aktuellen Entwicklungen Schritt zu halten und am aktuellen Stand zu bleiben. Gefordert sind hauptsächlich Entwickler, denn diese müssen dem Unternehmen bzw. dessen Führung die Vorteile der neuen Technologien schmackhaft machen (und sie selbt auch erlernen). Unternehmen, die die Möglichkeit bieten auf neue Technologien umzusteigen können daraus sicherlich Vorteile generieren. Dennoch ist der Faktor Mangelware an dieser Stelle Zeit. Zeit ist eine große Hürde die es zu überwinden gibt. Neue Technologien stellen ein große Herausforderung an das gesamte Unternehmen, vor allem an das Entwicklerteam, welches unter Zeitdruck steht und dennoch Erfolge bieten muss.

Umstiegskosten: Viele Unternehmen setzen auch heute noch Visual Studio 2003 ein. Dabei ist Visual Studio 2005 fast ein Jahr alt und die 2008er wartet bereits darauf fertig zu werden, um auf den Markt geworfen zu werden. Die Produktzyklen werden immer geringer und immer weniger machen den Schritt tatsächlich mit. Aus Kostengründen, denn Visual Studio ist nicht billig. Und WPF ist nun in der 2003er nicht verfügbar, vorhandene Projekte wollen nicht umgestellt werden, auf zwei Schienen entwickeln ist auch nicht unbedingt das Wahre usw.

Die Überwindung: Schließlich muss man sich überwinden, um tatsächlich ein Projekt auf Basis WPF durchzuziehen. Klar, wunderschön, man kann klar zwischen Designer und Entwickler trennen. Oft kann diese Trennung jedoch nicht vollzogen werden - aus Mangel an Designern. Allrounder sind also gefragt und diese müssen oft erst

dazu bewogen werden. Sind Designer vorhanden, müssen diese erst Grundlagen der WPF erlenen, denn ganz ohne geht es dann auch nicht.

Wahrscheinlich ließe sich diese Liste noch fortführen. Tatsache ist, dass es viele Gründe gibt, warum Unternehmen WPF nicht sofort einsetzen, sondern auf einen günstigen Moment warten, auf das richtige Projekt, die richtigen Entwickler, oder möglicherweise gar nie diesen Schritt wagen. Fakt ist, dass dieser Schritt irgendwann vollzogen werden sollte. Die Möglichkeiten sind groß, aber eine vorhandene Anwendung wird nun eben nicht von Heute auf Morgen umgestellt - wobei auch die Sinnhaftigkeit einer Umstellung hinterfragt werden sollte. Doch auch neue Projekte werden lieber mit Windows-Forms entwickelt. Und warum? Vermutlich weil es einfach an allgemein bekannten Anwendungen fehlt, die mittels WPF entwickelt wurden. Entsprechende Patterns sind ebenfalls Mangelware (dazu kommen wir in einem anderen Teil).

Was also sollte in meinem Unternehmen passieren um auf die neue Technologie zu kommen? Diese Frage ist immer schwer zu beantworten. Grundsätzlich sollte durch Visual Studio 2005 (bald 2008) die Grundlage gelegt werden. D.h. auf das .NET Framework 3.0 sollte umgestiegen werden. Diese muss nicht zwangsläufig für alle Anwendungen gelten. Es ist ok, sich nur eine kleine Anwendung (auf wenn diese nur für interne Verwaltungszwecke verwendet wird) heraus zu nehmen und diese quasi als Übung mit Hilfe der neuen Technologie umzusetzen. Mit den gewonnenen Erfahrungen kann man sich an größere Projekte wagen.

2.7. Windows Communication Foundation

2.7.1. Windows Communication Foundation: Ein paar Beispiele

Einer meiner kommenden Schwerpunkte wird auf jeden Fall die Windows Communication Foundation. Hier ein paar Links für all diejenigen, die sich weiter damit beschäftigen möchten:

- WCF Simple Example
- A truely simple example to get started with WCF
- WCF sample: Chat room client and server for .NET Framework 3.0
- WCF example: Using of self-hosted service with SSL

Zukünftig werden wohl auch einige Beispiele von mir erscheinen.

2.8. LINQ

2.8.1.LINQ to XML: Ein einfaches Beispiel

LINQ ist derzeit ein recht beliebtes Thema, viele Beiträge werden dazu verfasst. Auch ich möchte an dieser Stelle ein kleines, sehr einfaches Beispiel zu **LINQ to XML** zeigen. Es soll verdeutlichen, wie wenig Aufwand es bedarf, eine XML-Datei auszulesen. Zudem wird die Verwendung des **Schlüsselwortes var** gezeigt.

Ausgangspunkt ist das nachfolgende XML:

```
<?xml version="1.0" encoding="utf-8" ?>
<Persons>
  <Person>
    <Firstname>Norbert</Firstname>
    <Lastname>Eder</Lastname>
    <Email>csharp@gmx.at</Email>
    <Weblog>http://blog.norberteder.com</Weblog>
    <Active>1</Active>
  </Person>
  <Person>
    <Firstname>Hugo</Firstname>
    <Lastname>Tester</Lastname>
    <Email></Email>
    <Weblog></Weblog>
  </Person>
</Persons>
```

Und hier auch gleich das Beispiel:

```
if (File.Exists("Persons.xml"))
{
    XDocument personDoc = XDocument.Load("Persons.xml");
    var personList = from p in personDoc.Descendants("Person")
                     select new {
                         Firstname = (string) p.Element("Firstname"),
                         Lastname = (string) p.Element("Lastname"),
                         Email = (string) p.Element("Email"),
                         Weblog = (string) p.Element("Weblog"),
                         Active = (int?) p.Element("Active") ?? 0
                     };
    foreach (var person in personList)
    {
        Console.WriteLine("Firstname: " + person.Firstname);
        Console.WriteLine("Lastname:  " + person.Lastname);
        Console.WriteLine("Email:     " + person.Email);
        Console.WriteLine("Weblog:    " + person.Weblog);
        Console.WriteLine("Active:    " + person.Active.ToString());
        Console.WriteLine("----------------------------------------");
    }
    Console.ReadKey();
}
```

Grundsätzlich wird das XML in ein **XDocument** geladen. Nun wird mittels **var** ein anonymer Typ angelegt, welcher die Ergebnisliste der Abfrage aufnimmt.

Innerhalb des `Select new` werden die einzelnen Eigenschaften definiert und auf welches Element diese mappen. Per IntelliSense stehen in weiterer Folge diese Eigenschaften unseres anonymen Types zur Verfügung. Zu beachten ist an dieser Stelle vielleicht noch, dass die Eigenschaft `Active` als nullable Type definiert wird.

Dies hat den Hintergrund, dass es an dieser Stelle zu einer Exception käme, wäre das Element `Active` nicht vorhanden. Zusammen mit dem ??-Operator kann dadurch auf **null** abgefragt werden und in diesem Falle der Wert 0 zugewiesen werden.

Anschließend wird in einer `for each` Schleife sämtliche eingelesenen Datensätze ausgegeben. Und schon sind wir mit unserem Beispiel fertig.

2.8.2.Artikelserie LINQ

Die einen oder anderen werden das Blog von ScottGu kennen. Denjenigen, die es nicht kennen, sich jedoch mit LINQ anfreunden möchten, dem sei seine Artikelserie zum Thema **LINQ to SQL** ans Herz gelegt.

Part 1: Introduction to LINQ to SQL

Part 2: Defining our Data Model Classes

Part 3: Querying our Database

Part 4: Updating our Database

Part 5: Binding UI using the ASP:LinqDataSource Control

Part 6: Retrieving Data Using Stored Procedures

Part 7: Updating our Database using Stored Procedures

Part 8: Executing Custom SQL Expressions

Part 9: Using a Custom LINQ Expression with the control

Für den Einstieg und für die Schaffung von tiefergehendem Know-How ist diese Artikelserie absolut zu empfehlen.

2.9.Sonstiges

2.9.1.Erfolgsrückmeldung aus eigenen Anwendungen

Folgender Sachverhalt:
Eigene Anwendungen (vorzugsweise Konsolen-Anwendungen) werden im Batch ausgeführt (beispielsweise via NAnt, MSBuild etc.). Im Batch soll auf Abbruchbedingungen geprüft und entsprechend reagiert werden.

Wie kann dies einfach mit .NET 2.0 gelöst werden? Eigentlich ganz einfach. Wieder einmal hilft der `System.Environment-Namespace`. Mit Hilfe der Eigenschaft `ExitCode` besteht die Möglichkeit einen solchen zu setzen. 0 (Null) ist

standardmäßig gesetzt und bedeutet, dass die Anwendung erfolgreich ausgeführt wurde. Nun könnte es aber sein, dass es Rückmeldungen nicht nur für eine erfolgreiche Durchführung geben soll, sondern auch wenn Fehler oder Warnungen gemeldet werden sollen. Hierzu ist lediglich ein `Integer`-Wert zu definieren und `ExitCode` entsprechend zu setzen. Damit steht der Behandlung von Abbruchbedingungen im Batch nichts mehr im Weg.

2.9.2. Abhängigkeiten (Referenzen) führen zu Kompilations-Problemen

Wer kennt das nicht bei größeren Aufträgen bzw. Produkten: Mehrere Projekte tummeln sich in einer Solution und referenzieren einander. Oder noch schlimmer, die einzelnen Projekte sind in unterschiedlichen Solutions eingebunden. Und eventuell sind diese auch gleichzeitig geöffnet. Dann kann es zu folgender Fehlermeldung kommen:

```
Error: The dependency [name], Version=2.0.0000.0, Culture=neutral,
PublicKeyToken=[etc]' in project [name] cannot be copied to
the run directory because it would conflict with dependency [name],
Version=0.0.0000.0, Culture=neutral, PublicKeyToken=[etc]'.
```

In diesem Fall einfach das Visual Studio beenden und alles aus dem bin-Ordner des Projektes löschen. Danach das Projekt im VS laden und neu kompilieren. Nun sollte an sich wieder alles in Ordnung sein.

PS: Natürlich auch die einzelnen Referenzen überprüfen, ob es hier nicht notwendige Einträge in diversen Projekten der Solution gibt.

2.9.3. SVN Verzeichnisse mit einem Click entfernen

Der Subversion Client TortoiseSVN verwendet - wie viele andere auch - verstecke Ordner mit dem Namen .snv zur Verwaltung des aktuellen Zustands. Unter bestimmten Umständen ist es allerdings erforderlich diesen Ordner zu entfernen, um z.B. den Code weiterzugeben.

Es gibt nun mehrere Möglichkeiten. Entweder es wird die Windows Suche benutzt, alle Ordner per Hand entfernt oder die in TortouseSVN integrierte Export-Funktion benutzt. Noch einfacher ist es mit einem Eintrag in der Registry von Jon Galloway. Dieser Eintrag fügt dem Kontext-Menu des Explorers einen neuen Menüpunkt hinzun. Über diesen Menüpunkt ist es nun möglich, alle .svn Verzeichnisse mit nur einem Klick zu löschen.

Folgende Einträge müssen als .reg File abgespeichert und durch Doppelklick in die Registry importiert werden.

```
Windows Registry Editor Version 5.00
[HKEY_LOCAL_MACHINE\SOFTWARE\Classes\Folder\shell\DeleteSVN]
@="Delete SVN Folders"

[HKEY_LOCAL_MACHINE\SOFTWARE\Classes\Folder\shell\DeleteSVN\command]
@="cmd.exe /c \"TITLE Removing SVN Folders in %1 && COLOR 9A && FOR /r \"%1\" %%f
IN (.svn) DO RD /s /q \"%%f\" \""
```

Nutzer der TortoiseSVN Version für Visual Studio 2003 müssen den Namen des Verzeichnisses ".svn" durch "_svn" ersetzen.

3. Visual Studio

3.1. Auflistung Visual Studio Shortcuts

Kai hat sich die Mühe gemacht, die wichtigsten Visual Studio Shortcuts ein einem PDF zusammen zu fassen. Diese Liste ist sicherlich nicht vollständig, wird aber sehr wahrscheinlich um neue Shortcuts erweitert werden. Wer also zukünftig vermehrt auf seine Maus verzichten möchte, der sollte einen Blick darauf werfen.

3.2. Bookmarks aller geöffneten Dokumente ohne Rückfrage entfernen

Wer viel mit Bookmarks arbeitet der kommt auch des öfteren in die Lage, alle gesetzten Bookmarks entfernen zu wollen. Hierfür gibt es das Tastenkürzel CTRL + B + C. Allerdings erscheint hier die Nachfrage, ob denn wohl wirklich alle entfernt werden sollen. Wer dies nicht möchte, kann sich folgendes Makro einbauen und dann einem Button/Shortcut zuweisen - und das ohne diese Nachfrage. Dies funktioniert für alle geöffneten Dokumente:

```
Public Sub RemoveAllBookmarks()
Dim i As Integer

For i = 1 To DTE.Documents.Count
Dim doc As EnvDTE.TextDocument
doc = DTE.Documents.Item(i).Object
doc.ClearBookmarks()
Next
End Sub
```

3.3. Interessante VS 2005 Tastenkombinationen

Durch Zufall heute auf zwei interessante Tastenkombinationen des Visual Studios 2005 gekommen.

[CTRL] + [´]: Korrespondierende/s Klammer/Tag anspringen
[CTRL] + [']: Fokus in die Find-ComboBox setzen

Diese Tastenkombinationen funktionieren übrigens auch unter dem Visual Studio 2003.

3.4.Visual Studio 2005: Default Browser setzen

Ich hab es schon irgendwann einmal irgendwo gepostet, aber jetzt auf die Schnelle nicht gefunden, daher an dieser Stelle, damit ich es selbst nicht schon wieder vergesse.

Visual Studio 2005 öffnet gerne den Default-Browser fürs Debuggen und das muss natürlich nicht immer der Internet Explorer sein (oder man möchte explizit einen anderen Browser wählen). Dazu sind folgende Schritte notwendig.

Debugging stoppen, eine **ASPX-Datei** des Web-Projektes öffnen. Danach auf **File/Browse With ...** dadurch erscheint der folgende Dialog:

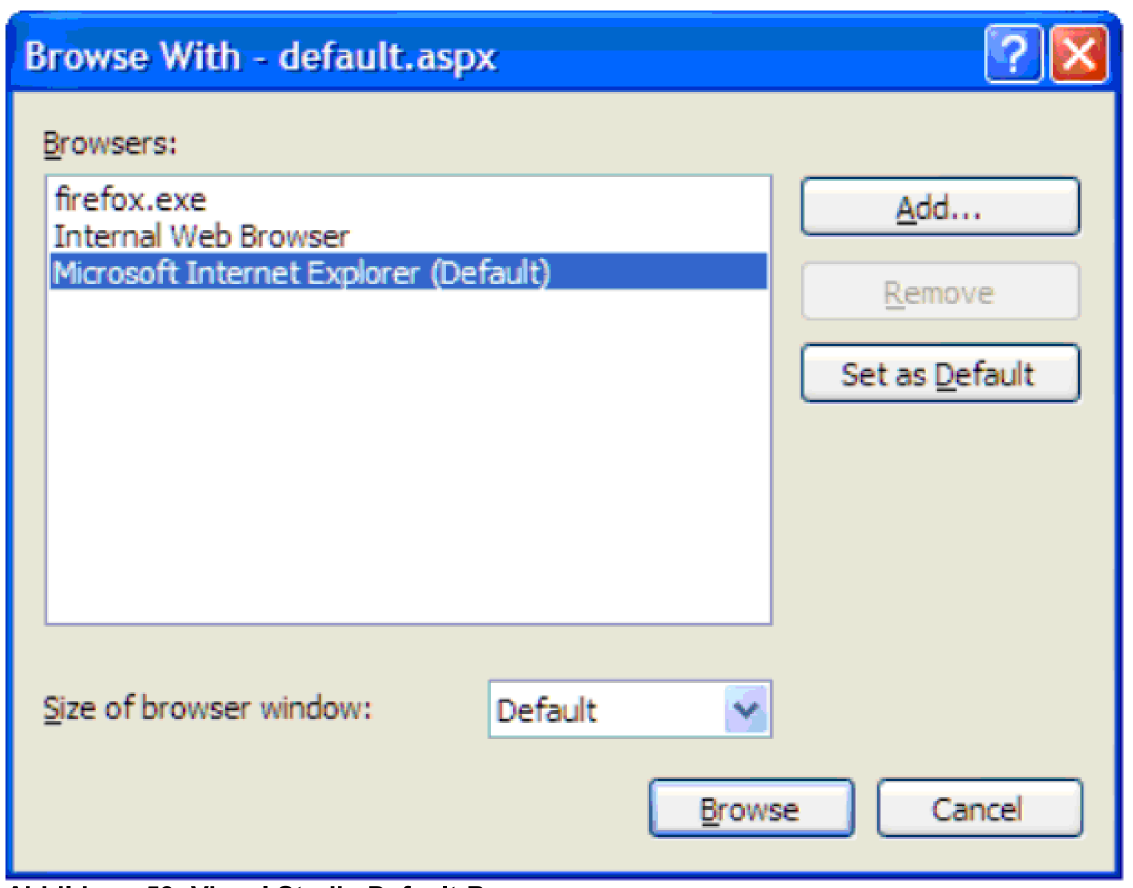


Abbildung 53: Visual Studio Default-Browser

Hier nun den gewünschten Browser auswählen, als Default setzen und auf **Browse** klicken. Ab sofort wird dieser Browser für die Debugging-Sessions verwendet.

3.5.Visual Studio: Anpassung Class Template

Ich persönlich unterteile meine Klassen gerne in Regionen nach privaten Feldern, Eigenschaften etc. um hier eine bestmögliche Übersicht zu erhalten. Natürlich ist es anstrengend, dies bei jeder neuen Klasse zu erstellen.

Um ein entsprechendes Ergebnis zu erhalten, kann das Class-Template angepasst werden. Aber gleich hier: Achtung!! Bei Fehlern kann ich nicht garantieren, wie sich das Visual Studio verhält - ich übernehme für etwaige Fehlfunktionen etc. keine Haftung.

So geht es
Benötigt wird die Datei *NewCSharpFile.cs*. Diese befindet sich im Programmverzeichnis des Visual Studios unter VC#\VC#\Wizards\CSharpAddClassWiz\Templates\1033.

Diese ist wie folgt anzupassen:

```
using System;

namespace [!output SAFE_NAMESPACE_NAME]
{
    /// <summary>
    /// Summary description for [!output SAFE_CLASS_NAME].
    /// </summary>
    public class [!output SAFE_CLASS_NAME]
    {
        #region Private Fields

        #endregion

        #region Properties

        #endregion

        #region Private Methods

        #endregion

        #region Public Methods

        #endregion

        #region ctor

        public [!output SAFE_CLASS_NAME]()
        {
            //
            // TODO: Add constructor logic here
            //
        }

        #endregion
    }
}
```

Abbildung 54: Visual Studio Class Template

Speichern und fertig. Ab sofort wird jede neue Klasse entsprechend dieses Templates erstellt. An dieser Stelle können natürlich auch weitere Informationen (Ersteller, etc.) angegeben werden.

3.6.Visual Studio 2005: Einfaches Einbinden von Namespaces

Wer sich schon immer darüber geärgert hat, dass Namespaces nicht automatisch eingebunden werden können (á la Eclipse), dem sei die Tastenkombination

<ALT><SHIFT><F10>

nahegelegt. Beispielsweise einfach mal `Hashtable` ins Codefenster schreiben, den Cursor gleich danach positionieren (also keine Leerzeichen eingeben) und die Tastenkombination ausführen. Schon kann man sich den richtigen Namespace aussuchen. Nettes Feature.

3.7.Visual Studio 2005: Taskbar einblenden

Wem eine Taskbar (ähnlich dem **<ALT><TAB>**) für Visual Studio 2005 abgeht, dem kann geholfen werden. Einfach mal die nachfolgende Tastenkombination ausprobieren:

<ALT><SHIFT><F7>

3.8.Visual Studio und Build Events

Oft müssen beim Erstellen eines Releases etc. Daten nachträglich in die Ausgabe-Ordner kopiert werden. Dies geht natürlich auch einfacher - automatisiert.

Dazu einfach die Projekteigenschaften öffnen (Menüpunkt Project / {Projectname} Properties).

Im darauf erscheinenden Fenster unter Common Properties / Build Events mit einem der beiden folgenden Einstellungen arbeiten:

- Pre-build Event Command Line
- Post-build Event Command Line

Wie der Name schon sagt, wird ersteres vor dem Build ausgeführt und zweiteres danach. Darin können nun die gewünschten Aktionen gesetzt werden. Beispielsweise das Kopieren von benötigten Dateien, die nicht ins Projekt eingebunden wurden etc. Einfach einmal auf den Erweiterungsbutton (der mit den 3 Punkten) klicken und im neuen Fenster auf "Macros >>". Darunter finden sich Eigenschaften die des Öfteren benötigt werden. Aufgerufen werden diese Eigenschaften mit $(Eigenschaftsname).

Im schlimmsten Falle einfach den Help-Button betätigen ;-)

3.9.Webprojekte mittels Firefox debuggen

Anstelle des Firefox können auch Opera etc. verwendet werden. Um den Browser umzustellen sind folgende Schritte notwendig:

In der geöffneten Web-Solution ein Webfile (beispielsweise eine apsx) wählen. Danach im Menü folgenden Eintrag wählen:

File / Browse with ...

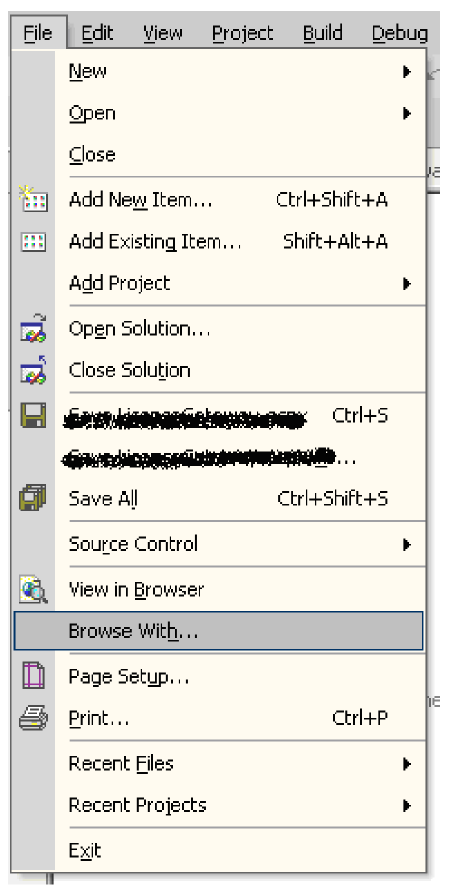


Abbildung 55: Webprojekte und Firefox 1

Im sich öffnenden Dialog einfach den gewünschten Browser auswählen und auf Default stellen.



Abbildung 56: Webprojekte und Firefox 2

Nun die Projekteigenschaften öffnen. Im Bereich

Configuration Properties / Debugging

gibt es den Eintrag 'Always Use Internet Explorer'. Dieser muss auf `false` stehen und schon ist die Umstellung komplett.

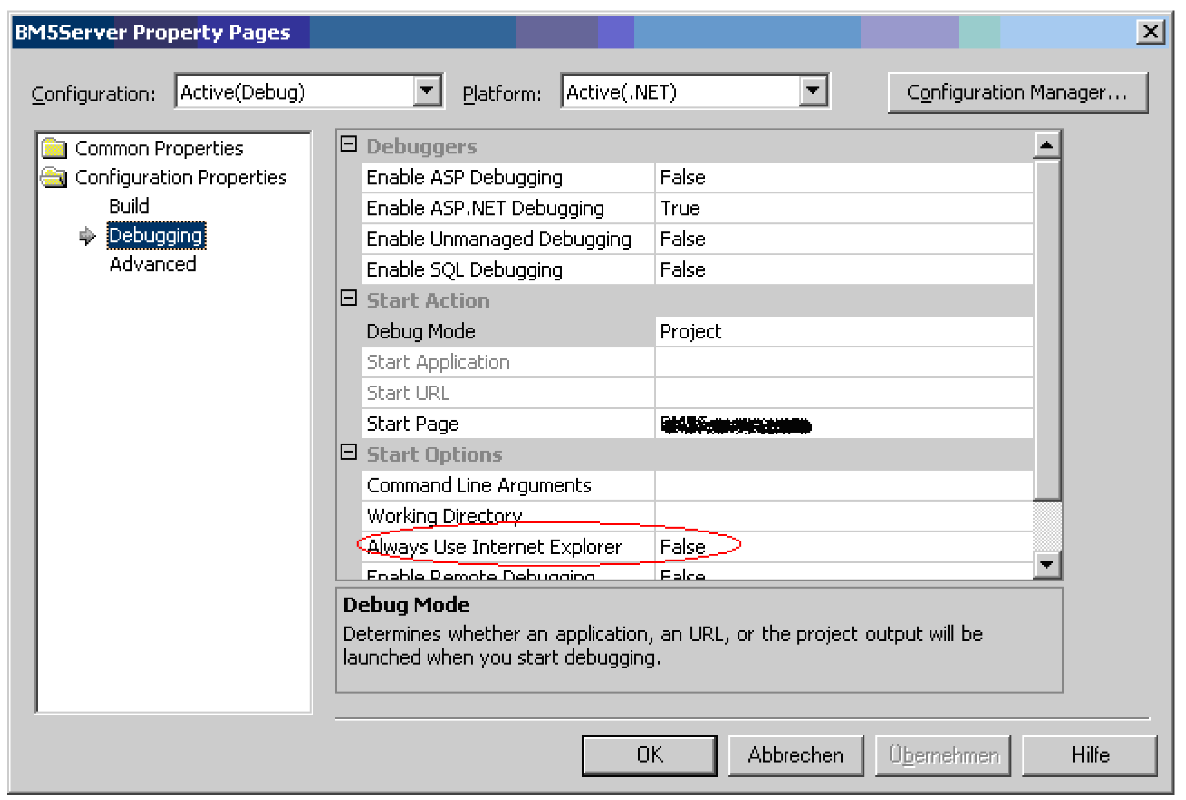


Abbildung 57: Webprojekte und Firefox 3

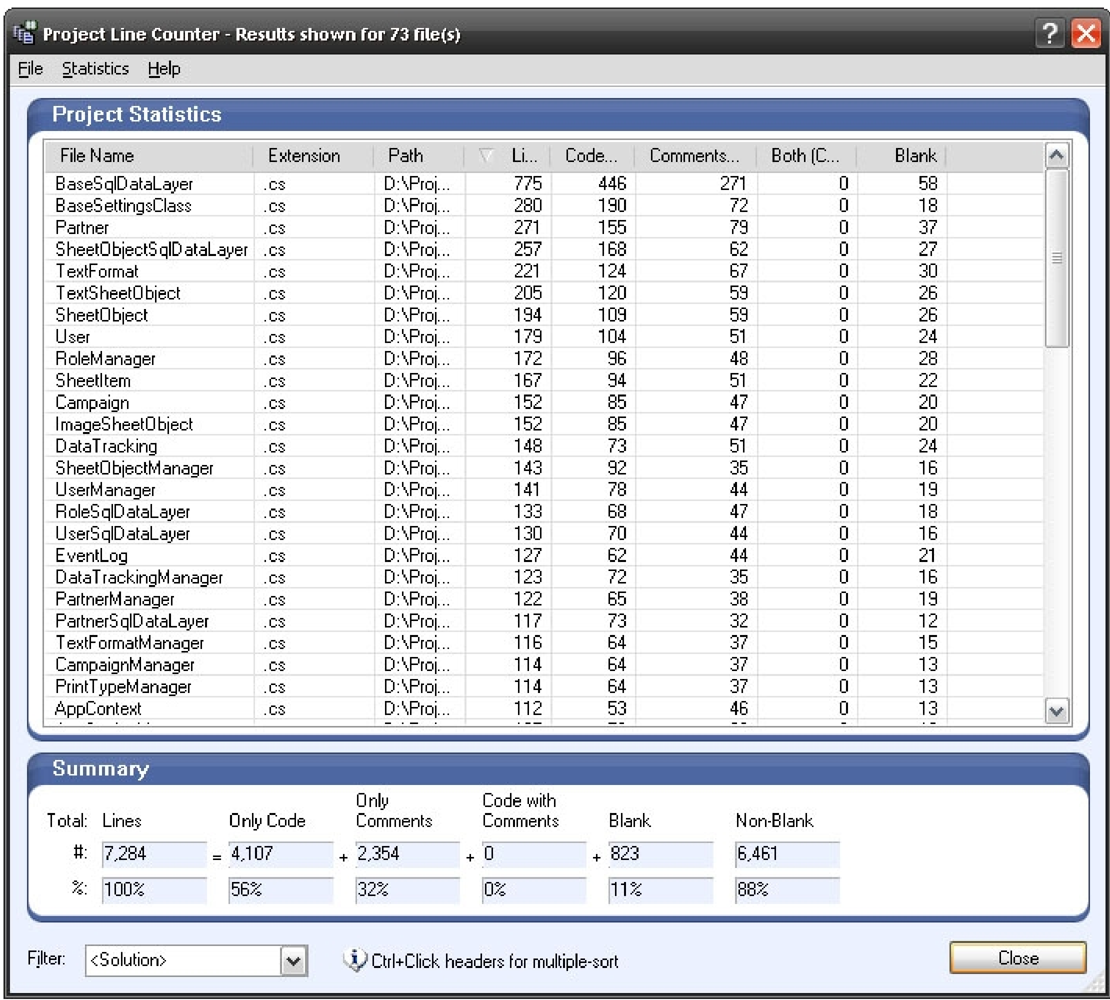
3.10.Codebehind Datei in Visual Studio 2005 hinzufügen

Der einfachste Weg um ein Codebehind/Codebeside File einer ASPX Seite hinzuzufügen, ist natürlich der entsprechende Schalter beim anlegen dieser Seite. Vergisst man dieses allerdings mal, oder möchte nachträglich noch eine Codebehind Datei hinzufügen, gibt es keine Möglichkeit dies mit einem entsprechenden Menüpunkt in Visual Studio zu erledigen.

Im Weblog von Jon Galloway findet man ein kleines Makro, welches genau dieses Job erledigt. Klick!

3.11.Project Line Counter für Visual Studio

Für ein Projekt benötigte ich ein paar Statistiken, wie z.B. Anzahl der Code oder Comment Zeilen. Nach kurzer Google Suche stieß ich auf das Tool *Line Counter* von Oz Solomon.



File Name	Extension	Path	Li...	Code...	Comments...	Both (C...	Blank
BaseSqlDataLayer	.cs	D:\Proj...	775	446	271	0	58
BaseSettingsClass	.cs	D:\Proj...	280	190	72	0	18
Partner	.cs	D:\Proj...	271	155	79	0	37
SheetObjectSqlDataLayer	.cs	D:\Proj...	257	168	62	0	27
TextFormat	.cs	D:\Proj...	221	124	67	0	30
TextSheetObject	.cs	D:\Proj...	205	120	59	0	26
SheetObject	.cs	D:\Proj...	194	109	59	0	26
User	.cs	D:\Proj...	179	104	51	0	24
RoleManager	.cs	D:\Proj...	172	96	48	0	28
SheetItem	.cs	D:\Proj...	167	94	51	0	22
Campaign	.cs	D:\Proj...	152	85	47	0	20
ImageSheetObject	.cs	D:\Proj...	152	85	47	0	20
DataTracking	.cs	D:\Proj...	148	73	51	0	24
SheetObjectManager	.cs	D:\Proj...	143	92	35	0	16
UserManager	.cs	D:\Proj...	141	78	44	0	19
RoleSqlDataLayer	.cs	D:\Proj...	133	68	47	0	18
UserSqlDataLayer	.cs	D:\Proj...	130	70	44	0	16
EventLog	.cs	D:\Proj...	127	62	44	0	21
DataTrackingManager	.cs	D:\Proj...	123	72	35	0	16
PartnerManager	.cs	D:\Proj...	122	65	38	0	19
PartnerSqlDataLayer	.cs	D:\Proj...	117	73	32	0	12
TextFormatManager	.cs	D:\Proj...	116	64	37	0	15
CampaignManager	.cs	D:\Proj...	114	64	37	0	13
PrintTypeManager	.cs	D:\Proj...	114	64	37	0	13
AppContext	.cs	D:\Proj...	112	53	46	0	13

Abbildung 58: Project Line Counter

Dieses Add-In für VS2003/VS2005 und VC++ 6.0 kann die Informationen entweder für die gesamte Solution oder jedes einzelne Projekt anzeigen. Zusätzlich ist es möglich die Ergebnisse als CSV oder XML Datei zu exportieren.

Zumindest für einen kurzen Überblick reicht das Tool völlig aus, zudem ist es kostenlos.

http://www.wndtabs.com/download/func,fileinfo/id,17/

3.12.Zur korrespondierenden Klammer springen

Durch Zufall bin ich heute auf einem weiteren sehr nützlichen Shortcut im Visual Studio gestoßen. Mit Hilfe der Tastenkombination *Ctrl* + ´ (siehe Abbildung) ist es möglich direkt zur korrespondierenden Klammer zu springen. Befindet sich der Cursor z.B. bei der schließenden Klammer einer Methode, springt er direkt zur öffnenden und somit zum Anfang der Methode.



Abbildung 59: Korrespondierende Klammer

Weitere nützliche Shortcuts findet man unter folgenden URLs:

http://www.codinghorror.com/Weblog/archives/000315.html
http://www.codeproject.com/tips/VSnetIDETipsAndTricks.asp
http://safari.oreilly.com/0596003609/mastvsnet-APP-C
http://Weblogs.dotnet-braunschweig.de/Karim/PermaLink,guid,2b2b0774-feba-4c88-9421-8cd44c4d7ed5.aspx

3.13.Visual Studio 2005 schneller starten

Heute bin ich zufällig über einen Eintrag von Peter Bucher gestolpert und diesen Tipp möchte ich natürlich meinen Lesern auch nicht vorenthalten.

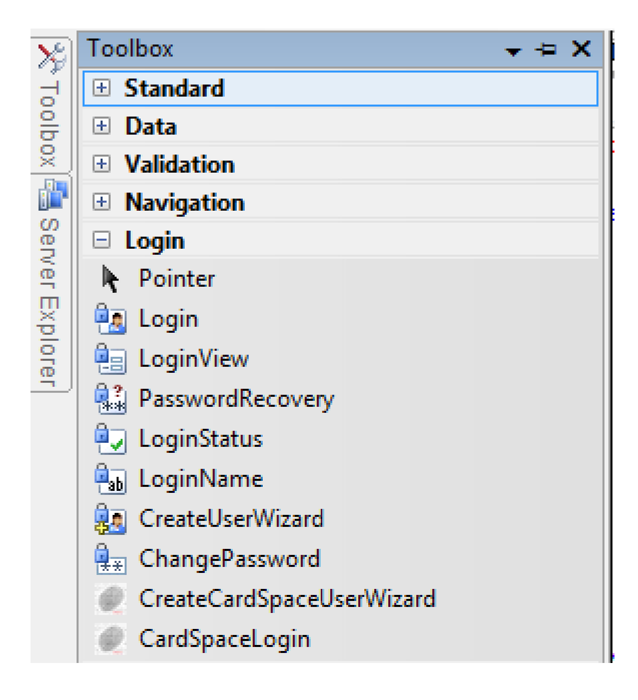


Abbildung 60: Visual Studio schneller starten

In der Verknüpfung von Visual Studio 2005 einfach ein **/nosplash** anfügen und schon startet Visual Studio 2005 um ein paar Sekunden schneller.

3.14. Visual Studio 2005 Toolbox für Windows CardSpace

Christian Arnold hat eine Visual Studio 2005 Toolbox speziell für die Arbeit mit Windows CardSpace geschrieben. Mit Hilfe dieser zwei neuen Controls mit dem Namen `CardSpaceLogin` und `CreateCardSpaceUserWizard`, ist es auf einfache Weise möglich Windows CardSpace in eigenen ASP.NET Anwendungen zu benutzen.



Mehr Informationen, eine Installationsanleitung und natürlich den Download gibt es hier.

3.15.Neue Visual Studio 2005 Code Snippets für Office 2007

Microsoft hat neue Visual Studio Code Snippets veröffentlicht. Diese Snippets sind speziell für Projekte in Verbindung mit Office 2007 und sind in C# und VB.NET verfügbar.

Mehr Informationen und natürlich den Download gibt es hier.

3.16.Visual Studio 2008: Bessere Performance?

Gerade das Thema Performance war punkto Visual Studio immer ein Spezialfall. Sobald Solutions wachsen wird auch die Entwicklungsumgebung immer langsamer. Zeit also, hier einige Verbesserungen zu bringen.

Soma Somasegar führt auf, an welchen Stellen gedreht wurde um die Performance von Visual Studio 2008 zu verbessern. Hier ein Auszug:

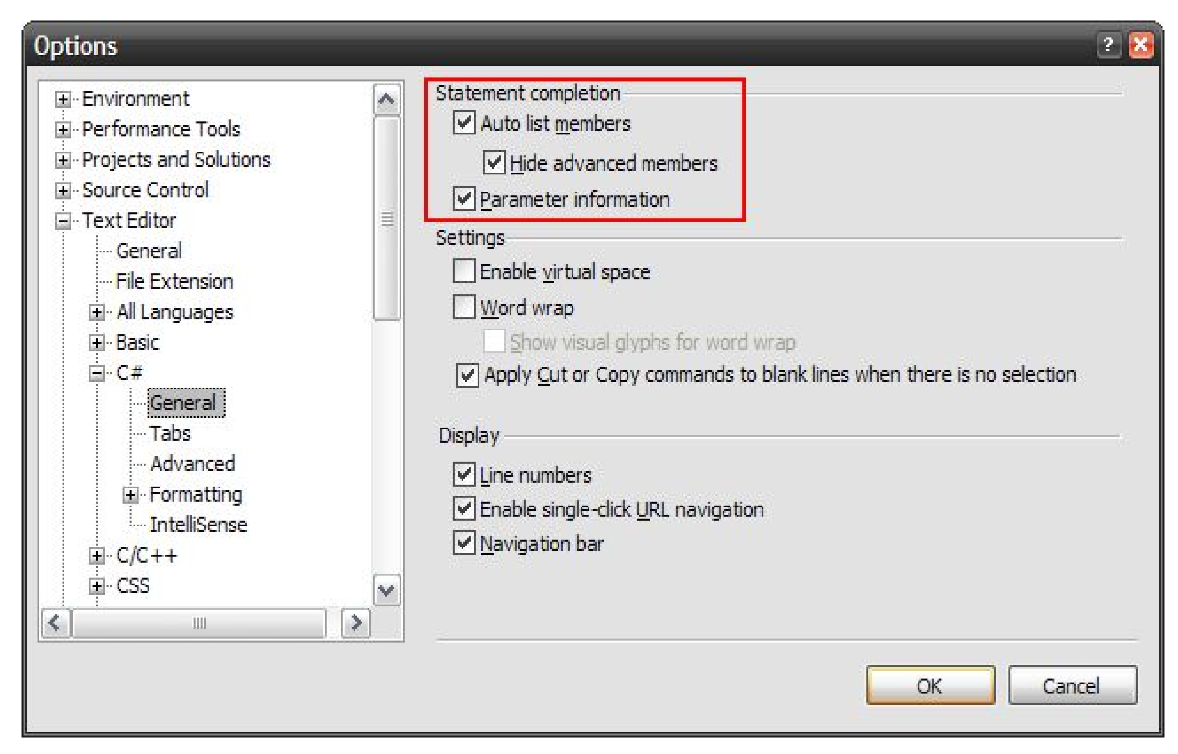
- Rebuilding a Visual Basic project and running a background compiler is 3x faster and uses 3x less memory.
- Scrolling large C# files in the Editor is 100% faster, while typing in new text is 50% faster
- The response time of IntelliSense with large types in C# is up to 10 times faster
- Incremental build time runs up to 90% faster on C++/CLI project solutions.
- Office Word and Excel files are processed 20x faster on the server
- TFS Version Control command processing was re-written to support unlimited sized operations on key commands without being memory bound on the server. In our measurements, key Commands also run 10% - 60% faster, with the larger improvements associated with bigger projects.

3.17.Visual Studio 2005: IntelliSense funktioniert nicht mehr

Wer sich beispielsweise den ReSharper testhalber installiert, könnte nach der Deinstallation bemerken, dass IntelliSense nicht mehr funktioniert. Hierzu sind folgende Einstellungen zu treffen, damit es wieder verwendet werden kann:

Unter **Tools - Options** auf **Text Editor - C# - General** müssen folgende Einträge aktiviert sein:

- Auto list members
- Parameter information



Danach sollte IntelliSense wieder verfügbar sein. Ein anderer Weg könnte ebenfalls zielführend sein: Vor der Deinstallation des ReSharpers in dessen Optionen IntelliSense wieder zurück an Visual Studio geben. Vermutlich sollten dann die oben angeführten Einstellungen nicht mehr getroffen werden.

Gewinncode: 27AC97U

3.18.Visual Studio 2005: Refactoring-Performance verbessern

Mit zunehmender Projektgröße wird auch das integrierte Refactoring ständig langsamer, bis es schließlich überhaupt nicht mehr verwendbar ist. Äußerst schlimm ist dies bei Web Projekten. Hintergrund ist der, dass so gut wie jeglicher Content für das Refactoring herangezogen wird. Eine Möglichkeit, einzelne Projekte auszuwählen oder bestimmte Teile der Solution nicht mit einzubeziehen ist ebenfalls nicht vorhanden. Dennoch gibt es eine kleine Variante, welche das Refactoring deutlich beschleunigt. Wie in vielen Fällen ist die Lösung über die Registry möglich:

1. Alle offenen Visual Studio 2005 Instanzen schließen

2. **regedit** starten
3. HKEY_CURRENT_USER\Software\Microsoft\VisualStudio\8.0\Csharp\Options\Editor
4. Neuer Eintrag: "Neuer DWORD-Wert"
5. Dem Eintrag den Namen **OpenAllVenusFilesOnRefactor** verpassen und den Wert auf 0 belassen

Nach einem Neustart sollte nun das Refactoring deutlich schneller laufen, ASP.NET Pages, Inline .asmx und .asax Code werden nicht mehr berücksichtigt.

Wer die Startzeit von Visual Studio 2005 verbessern möchte, sei auf den Eintrag Visual Studio 2005 schneller starten verwiesen.

Und noch ein Hinweis: Refactoring gibt es auch für Visual Basic 2005. Hierfür einfach Refactor! verwenden.

3.19.Visual Studio 2005: Region-Blöcke per Tastatur auf- und zuklappen

Regions sind eine sehr nette Sache, möchte man Methoden, Eigenschaften, Attribute oder andere zusammengehörigen Teile einer Datei in Bereiche unterteilen. Wer nun aber möglichst wenig zur Maus greifen möchte, stellt sich sehr schnell die Frage, wie denn Regions-Blöcke per Tastatur auf- bzw. zugeklappt werden können:

STRG+M STRG+M

Und schon wieder ein Griff weniger zur Maus. Dass es doch tatsächlich immer wieder Tastenkombinationen gibt, die man ständig vergisst ...

3.20.Visual Studio 2005: Anpassung Class Template

Im Beitrag Visual Studio: Anpassung Class Templates habe ich bereits beschrieben, wie die Klassen-Templates (wird verwendet beim Anlegen einer neuen Klasse) an eigene Bedürfnisse angepasst werden können. Hier nun, wie dies unter Visual Studio 2005 gemacht werden kann.

Die Class-Templates können unter

```
C:\Programme\Microsoft Visual Studio
8\Common7\IDE\ItemTemplatesCache\CSharp\1033\Class.zip\
```

gefunden werden (liegen also in einem anderen Pfad, als unter Visual Studio 2003). Im darüberliegenden Verzeichnis finden sich übrigens die Templates für die AssemblyInfo, Form, den Installer und alle anderen bekannten Templates. Diese sind

weiters in die jeweiligen Programmiersprachen unterteilt, wodurch auch VB.NET-Entwickler ihre Templates schnell finden sollten.

Standardmäßig sieht das Class-Template so aus:

```
using System;
using System.Collections.Generic;
using System.Text;
namespace $rootnamespace$
{
    class $safeitemrootname$
    {
    }
}
```

So manchem stört das fehlende **public**, sowie auch - in einigen Fällen - vordefinierte **Regions**. Durch folgende Anpassung der Vorlage gehört dieser Missstand der Vergangenheit an:

```
using System;
using System.Collections.Generic;
using System.Text;
namespace $rootnamespace$
{
    public class $safeitemrootname$
    {
    #region Attributes

    #endregion

    #region Properties

    #endregion

    #region Private Methods

    #endregion

    #region Public Methods

    #endregion
    }
}
```

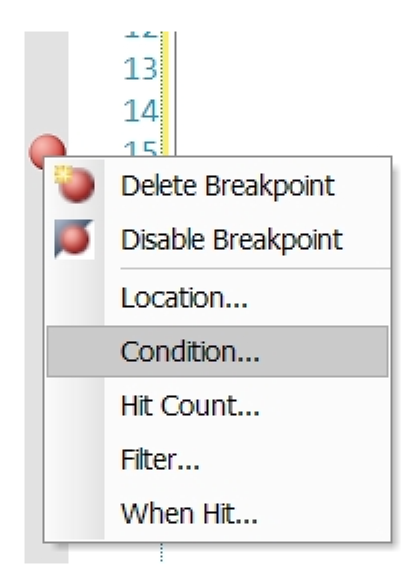
So einfach wie diese Erweiterung können auch zusätzliche Namespaces eingetragen werden.

3.21.Tipps zum Debugging

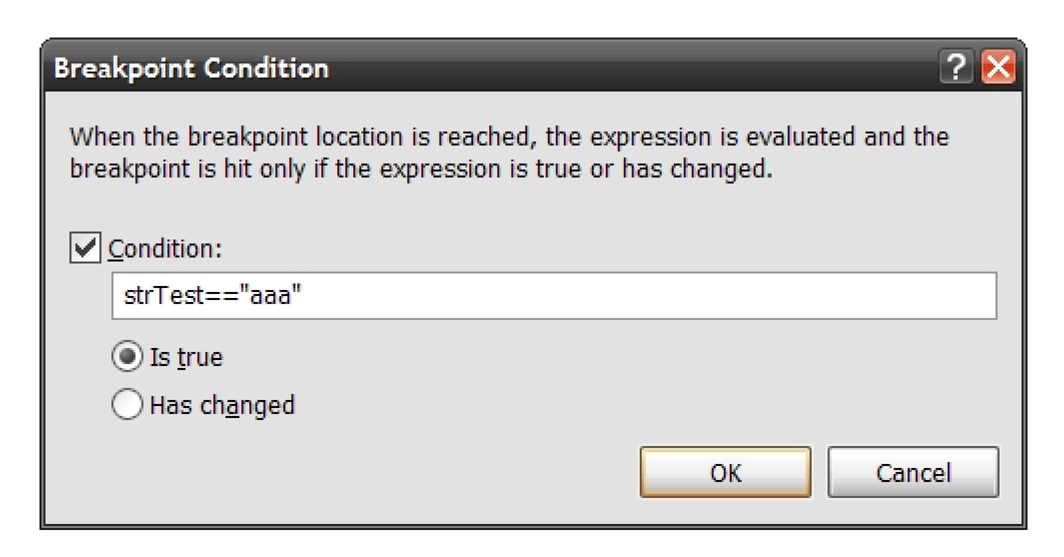
Visual Studio 2005 bietet kleine nette Features zum Thema Debuggen an, die nicht sehr häufig von Entwicklern genutzt werden. Dabei können diese kleinen Möglichkeiten das Leben durchaus erleichtern.

Setzen wir an einer bestimmten Stelle im Sourcecode (ich verwende hier ein paar

wenige Sourcecode-Zeilen zur Veranschaulichung). Per rechter Maustaste auf den Breakpoint erhalten wir ein Kontextmenü mit einigen hilfreichen Einträgen.



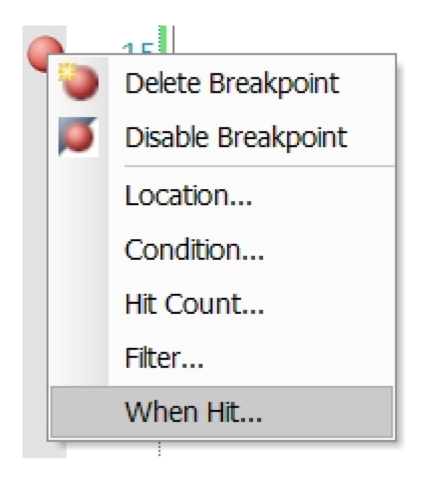
Sehen wir uns den Punkt **Condition...** genauer an.



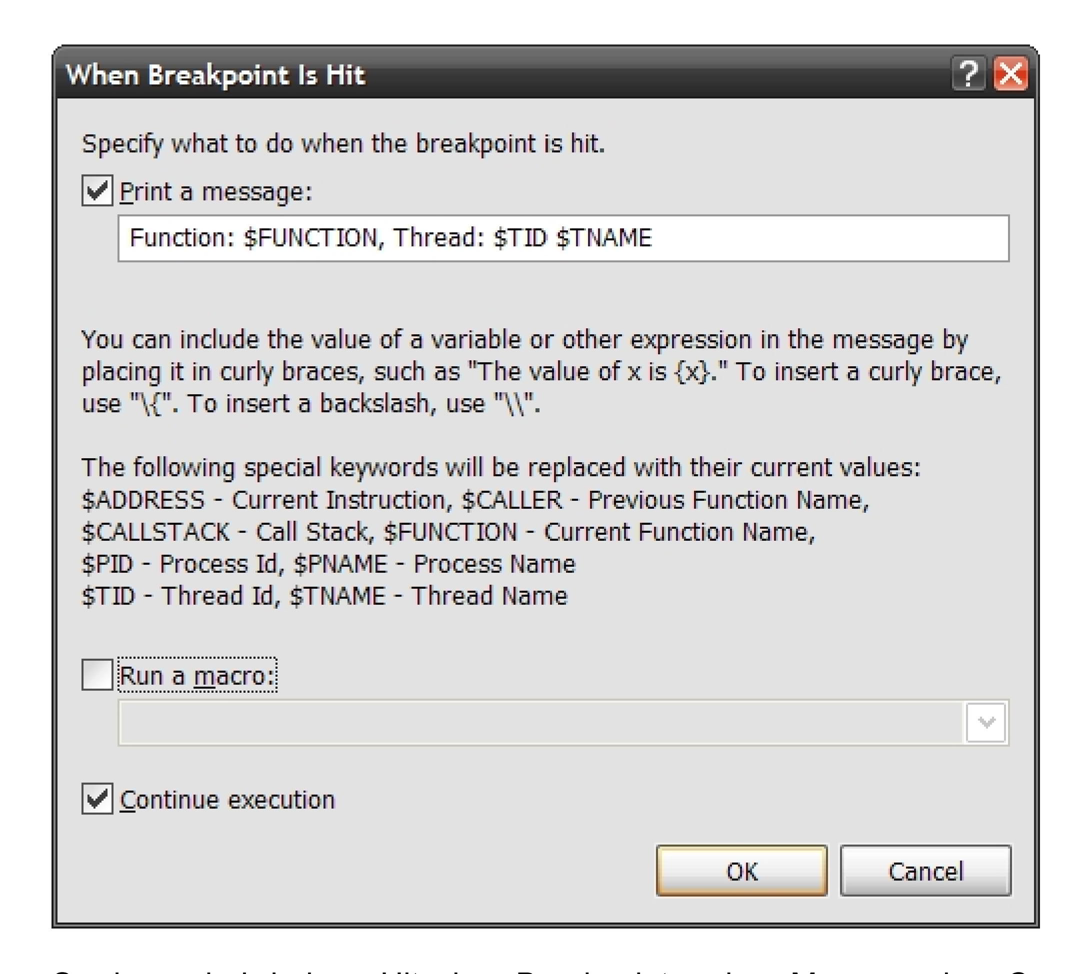
Beim Durchlaufen des Sources wird bei jedem Hit des Breakpoints die eingegebene Bedingung überprüft. Trifft diese zu (sofern der Punkt **Is true** gewählt wurde), wird an dieser Stelle unterbrochen, wie im nächsten Screenshot gut zu sehen ist.

```
 7  class Program
 8  {
 9      static void Main(string[] args)
10      {
11          string strTest = String.Empty;
12
13          for (int i = 0; i < 10; i++)
14          {
15              strTest += "a";
16          }
17      }
18  }
```

Eine weitere Möglichkeit ist durch den Punkt **When Hit...** gegeben.



Durch diesen wird ein Einstellungsfenster geöffnet, in dem einige Settings vorgenommen werden können:



So kann bei jedem Hit des Breakpoints eine Message ins Consolen-Fenster geschrieben, ein Makro ausgeführt, oder unterbrochen werden.

Es lohnt sich auf jeden Fall, auch die anderen Punkte näher anzusehen, da bestimmte Features recht oft sinnvoll eingesetzt werden können. Damit läßt sich die Produktivität steigern und der Ärger des Debuggens senken.

Weitere Informationen zum Thema Debuggen:

Visual Studio 2005: Default Browser setzen

Webprojekte mittels Firefox debuggen

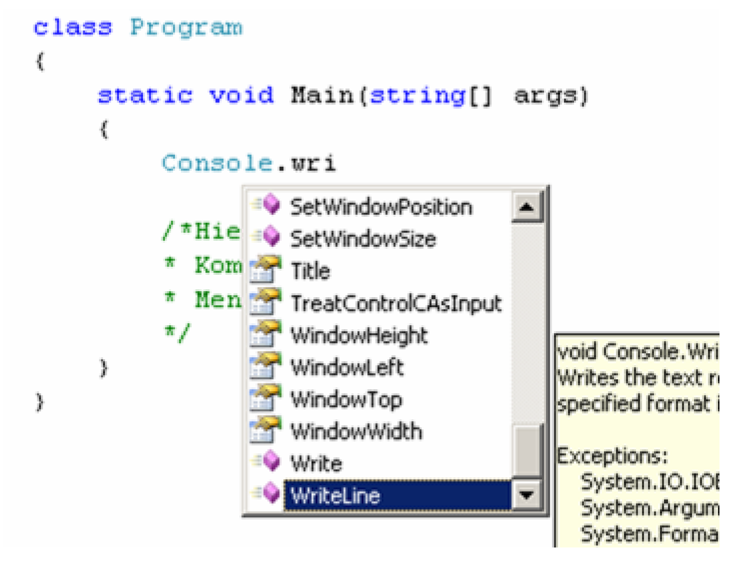
Unable to debug: The binding handle is invalid.

3.22. Transparentes IntelliSense in Visual Studio 2008

Während der Arbeit mit Visual Studio 2008 fallen immer wieder kleine, aber dennoch sehr nützliche Hilfen auf. So auch die Möglichkeit das IntelliSense transparent zu schalten.

Das Problem, welches dieser Lösung voraus geht, sollte vielen Visual Studio Nutzer

bekannt vorkommen. Man schreibt Quellcode und plötzlich wird eine wichtige Stelle durch das IntelliSense verdeckt. Allerdings benötigt der Entwickler genau diesen Teil oder Ausschnitt um die begonnene Zeile zu Ende zu schreiben.



Nach dem Drücken der Strg-Taste, wird das IntelliSense transparent gemacht und der versteckte Quellcode kommt zum Vorschein.

```
class Program
{
    static void Main(string[] args)
    {
        Console.wri

        /*Hier stehen
        * Kommentare oder jede
        * Menge Quellcode
        */
    }
}
```

Obwohl Visual Studio viele solcher kleinen Helferlein besitzt, wissen die wenigsten Entwickler davon. Es lohnt sich also sich auch mal mit der Entwicklungsumgebung selbst und nicht nur mit der Programmiersprache oder Technologie auseinanderzusetzen.

3.23.Silverlight 1.0 JavaScript IntelliSense

Visual Studio 2008 wird mit IntelliSense für JavaScript ausgeliefert. Damit das IntelliSense für Silverlight 1.0 Objekte und Methoden funktioniert hat Justin-Josef Angel eine Hilfsbibliothek geschrieben und das Ganze auf CodePlex veröffentlicht.

```
function play_Clicked(sender, args)
{
    var play = Convert.ToCanvas(sender);

    play.fin
    se  add_MouseMove                nt").Play();
        captureMouse
}       constructor
        equals
        findName       Gets any object in the Silverlight object hierarchy by referencing the object's x:Name attribute value.
functi  get_background               rgs)
{       get_children
    al  get_clip
    se  get_cursor                   nt").Stop();
}       get_height
```

Die Installation bzw. Einrichtung ist denkbar einfach und ebenfalls in der ReadMe.txt des Downloadarchivs beschrieben. Zunächst müssen die beiden Dateien *intellisense.compressed.js* und *intellisense.js* in den Projektordner entpackt werden. Um die Bibliothek in JavaScript Dateien zu referenzieren genüg es einen *reference* Hinweis am Anfang der Datei einzufügen.

```
///
   <reference path="intellisense.js" />
function play_Clicked(sender,
   args)
{
 var bttPlay
   = Convert.ToCanvas(sender);
 bttPlay.findName("audioElement").Play();
}
```

Zusätzlich muss in der Html Datei, die das Silverlight Objekt darstellt, eine weitere Referenz hinzugefügt werden.

```
<script type="text/javascript" src="intellisense.compressed.js"></script>
```

Das Ergebnis ist ein funktionierendes JavaScript IntelliSense und Debugging für Silverlight 1.0. Auf der Codeplex Projektseite findet man dazu nicht nur den Download, sondern auch ein Video, welches diese Schritte noch mal ausführlich erklärt und aufzeigt.

http://www.codeplex.com/intellisense

3.24.CopySourceAsHtml mit Visual Studio 2008 Beta 2 nutzen

Bereits seit mehreren Jahren nutze ich das Visual Studio AddIn CopySourceAsHtml um Sourcecode in Html umzuwandeln. Leider gibt es momentan noch keine Visual Studio 2008 Version. Trotzdem ist es möglich das AddIn zu nutzen.

Zunächst müssen alle zugehörigen Dateien aus dem Visual Studio 2005 AddIns Ordner in den Visual Studio 2008 Add-Ins Ordner kopiert werden. Unter Windows XP befinden sich die Dateien im Pfad

```
C:\Dokumente und Einstellungen\<BENUTZER>\Eigene Dateien\Visual Studio
<2005><20008>\AddIns

CopySourceAsHtml.AddIn
CopySourceAsHtml.dll
CopySourceAsHtml.dll.config
CopySourceAsHtml.pdb
```

Nun muss die angegebene Visual Studio Version in der Datei *CopySourceAsHtml.AddIn* auf 9.0 geändert werden.

```
<?xml version="1.0" encoding="UTF-16" standalone="no"?>
<Extensibility xmlns="http://schemas.microsoft.com/AutomationExtensibility">
 <HostApplication>
  <Name>Microsoft
   Visual Studio Macros</Name>
  <Version>9.0</Version>
 </HostApplication>
 <HostApplication>
  <Name>Microsoft
   Visual Studio</Name>
  <Version>9.0</Version>
 </HostApplication>
 <Addin>
  <FriendlyName>CopySourceAsHtml</FriendlyName>
  <Description>Adds
   support to Microsoft Visual Studio 2005
for copying source code, syntax highlighting, and line numbers as
HTML.</Description>
  <Assembly>CopySourceAsHtml.dll</Assembly>
  <FullClassName>JTLeigh.Tools.CopySourceAsHtml.Connect</FullClassName>
  <LoadBehavior>1</LoadBehavior>
  <CommandPreload>0</CommandPreload>
  <CommandLineSafe>0</CommandLineSafe>
 </Addin>
</Extensibility>
```

Anschließend kann über den Add-In Manager das Tool aktiviert und sofort verwendet werden.

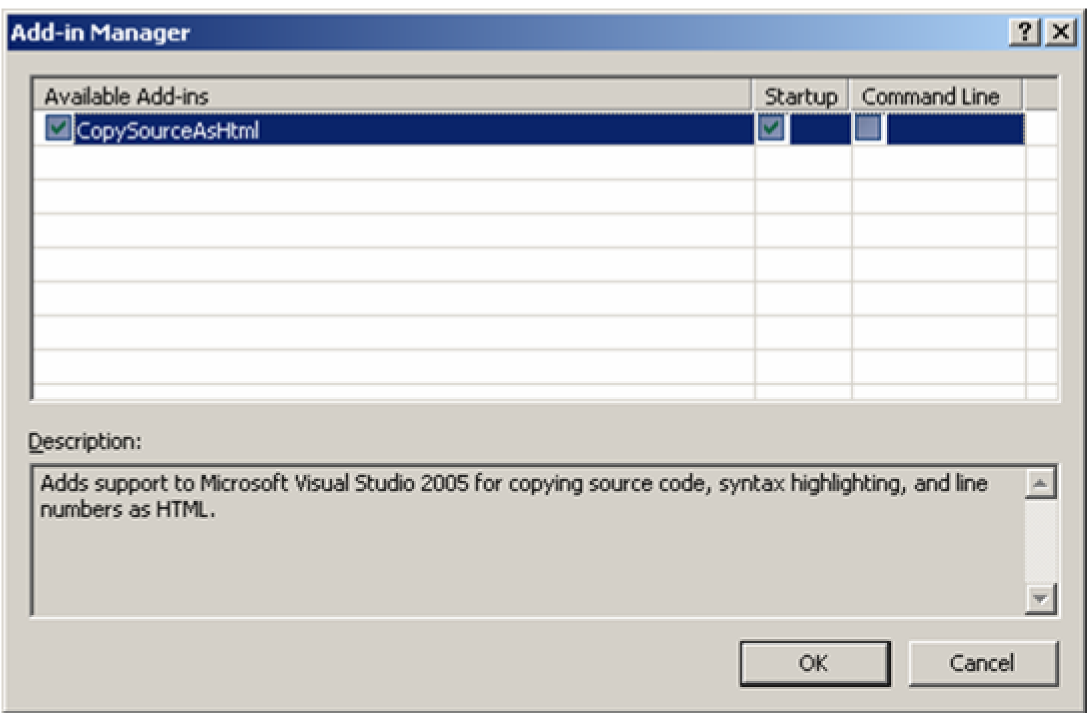
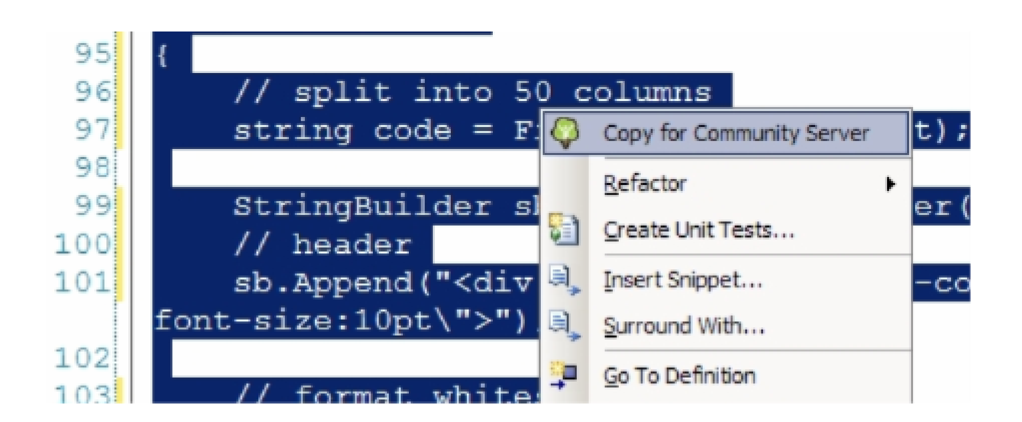


Abbildung 61: Visual Studio Add-In Manager

3.25.Visual Studio Plugin für Community Server

Wer häufig Code Snippets in den Community Server Blogs oder Foren postet, wird sich über das Visual Studio Plugin von Zeddy Iskandar freuen. Dies erlaubt ein einfaches Copy & Pase von Quellcode in den Community Server.



Mehr Informationen gibt es unter

http://geeks.netindonesia.net/blogs/zeddy/archive/2007/04/25/Visual-Studio-Plugin-for-Community-Server.aspx

4. Entwicklung

4.1. Allgemein

4.1.1. Continuous Integration - kenn ich nich ..

Nein? Dann dürfte ein Artikel von Martin Fowler [1] zu diesem Thema [2] wirklich weiterhelfen.

Continuous Integration is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily - leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible. Many teams find that this approach leads to significantly reduced integration problems and allows a team to develop cohesive software more rapidly. This article is a quick overview of Continuous Integration summarizing the technique and its current usage.

[1] Martin Fowler
[2] Continuous Integration

4.1.2. Überladen vs. überschreiben

Da ich heute wieder auf eine Verwechslung dieser beiden Begriffe (engl. Override, Overload) gestoßen bin, möchte ich diese kurz hier erklären.

Überladung

Beim Überladen wird die Signatur einer Methode verändert. Der Methodenname ändert sich jedoch nicht. Was sich hierbei ändert sind die Parameter bzw. die Anzahl der Parameter. Der Rückgabetyp kann sich ändern, jedoch müssen die Parameter unterschiedliche Typen aufweisen, oder die Anzahl der Parameter muss unterschiedlich sein.

Was ist die Signatur?

Unter der Signatur versteht man die Definition einer Methode, sprich mehr oder weniger der Methoden-Rumpf.

Beispiel:

```
public string CalcValue(int value1, int value2) {}
```

Was darf hier nun geändert werden? Wie gesagt, der Methoden-Name muss ident bleiben. Eine Änderung des obigen Beispiels zu

```
public string CalcValue(int value1, double value2) {}
```

wäre erlaubt. Dies bedeutet nun, dass sich auch die Typen der Parameter ändern können. Besteht allerdings nur ein Parameter vom Typ int namens value1, dann darf der es nicht sein, dass der Typ derselbe bleibt, sich jedoch der Name ändert. Diese Möglichkeit besteht nicht.

Überschreibung

Beim Überschreiben einer Methode sieht dies anders aus. In diesem Fall bleibt die Signatur dieselbe. Das heißt, weder der Rückgabewert, noch der Methodenname, noch die Parameter (auch nicht deren Typen, Parameternamen, Anzahl der Parameter) ändern sich. Hier ändert sich der Innenteil der Methode, sprich die Funktionalität.

Beispielsweise wird

```
public virtual int Calc(int value1, int value2)
{
    return value1 + value2;
}
```

zu

```
public override int Calc(int value1, int value2)
{
    return value1 * value2;
}
```

Einsatzgebiete

Überladung wird beispielsweise verwendet, wenn eine Methode mehrere optionale Parameter besitzen soll bzw. unterschiedliche Eingangswerte haben soll, um Grunde aber entsprechend der Parameter eine entsprechende Aufgabe löst.

Überschreibung kommt bei der Vererbung zu tragen. So kann eine Basisklasse die Implementierung einer Methode vorschreiben, die jedoch bei jeder Ableitung eine unterschiedliche Aufgabe besitzt, aufgrund der weiteren Verwendung jedoch immer die gleiche Signatur besitzen muss.

4.1.3.Kopier-Konstruktor in C#

Unter vielen Sprachen ist ein Kopier-Konstruktor standardmäßig für ein Objekt vorhanden. Bei C# ist dem nicht so. Nachfolgend möchte ich kurz zeigen, wie ein Kopier-Konstruktor in C# realisiert wird. Im Anschluss gibt es einen Hinweis von mir, warum ich persönlich Kopier-Konstruktoren nicht verwenden würde.

```
public class Car
```

```
{
    private int _maxSpeed = 200;
    private Color _color = Color.White;
    private Driver _driver = null;

    public int MaxSpeed
    {
        get { return this._maxSpeed; }
        set { this._maxSpeed = value; }
    }

    public Color Color
    {
        get { return this._color; }
        set { this._color = value; }
    }

    public Driver Driver
    {
        get { return this._driver; }
        set { this._driver = value; }
    }

    public Car() { }

    public Car(Car car)
    {
        this._color = car.Color;
        this._maxSpeed = car.MaxSpeed;
        Driver d = new Driver();
        d.Firstname = car.Driver.Firstname;
        d.Lastname = car.Driver.Lastname;

        this._driver = d;
    }
}
```

Es gibt eine Überladung des Konstruktors, welche ein Objekt vom Typ `Car` entgegen nimmt. Hierzu wird ein Deep Copy durchgeführt und die Daten entsprechend kopiert.

Dies klingt ja prinzipiell recht einfach und auch ganz gut. Was daran passt also nicht? Nun, man gehe von der Verwendung in einem Framework, welches anderen Entwicklern zur Verfügung gestellt wird aus, oder davon, dass sich der eigentliche Entwickler einige Wochen später nochmals an dieses Projekt setzt. Hier sieht er beispielsweise folgenden Code:

```
Car c = new Car();
Driver d = new Driver();
d.Firstname = "Norbert";
d.Lastname = "Eder";
c.MaxSpeed = 250;
c.Color = Color.Blue;
c.Driver = d;

Car c2 = new Car(c);
```

Auf den ersten Blick ist nicht ersichtlich, dass die Daten aus dem Objekt in das neu zu erstellende Objekt kopiert werden. Hierzu ist ein Blick in die Klasse notwendig. Ein Schritt, der nicht sein muss. Stattdessen empfiehlt es sich, das Interface `ICloneable` zu verwenden, oder mit `MemberwiseClone` zu arbeiten. Je nachdem welche Kopiermethode verwendet werden soll.

4.1.4.Serviceorientierte Architekturen Grundlagen

Serviceorientierte Architektur (SOA) ist wohl ein häufig gebrauchter Terminus in der heutigen Zeit. Wer sich beispielsweise näher mit der **Windows Communication Foundation** (WCF) beschäftigen möchte, sollte sich zuvor in die Grundlagen der SOA einarbeiten.

Wie die Bezeichnung vermuten lässt, besteht SOA aus lose gekoppelten Diensten, die jeweils bestimmte Aufgaben kapseln und unabhängig voneinander bezogen werden können. Ein Dienst wird von einem **Service Provider** angeboten, der Client nennt sich **Service Consumer**.

Ein einzelner Dienst stellt unterschiedliche Funktionen (Operationen) zur Verfügung, die von einem Consumer aufgerufen werden können. Dabei wird ein Service über eine Schnittstelle (Contract) definiert. Die Schnittstelle beschreibt also, welche Funktionalitäten und Nachrichten angeboten werden. Hierbei ist zusätzlich darauf zu achten, dass der Client die Implementierung der Funktionalität nicht kennt, da diese vom Service gekapselt wird. Dadurch ist es sehr einfach möglich, die Funktionalität selbst zu ändern, ohne Änderungen am Client (und ein damit verbundenes Rollout) vornehmen zu müssen.

Nun stellt sich die Frage, über welches Protokoll kommuniziert wird. Dies wird in so genannten **Policies** festgelegt.

Wie erfolgt nun der Ablauf der Kommunikation?

Die Kommunikation erfolgt prinzipiell durch das Versenden und Empfangen von Nachrichten. Diese enthalten Daten und keine Objekte. Der Austausch erfolgt über so genannte **Endpoints** die vom jeweiligen Service bereitgestellt werden. Ein Endpoint besteht aus drei Teilen:

- Adresse
- Binding
- Kontrakt

Die **Adresse** definiert, wo der Endpoint zu finden ist. Das **Binding** beschreibt, wie der Endpoint aufgerufen wird und der **Kontrakt** (wie bereits weiter oben beschrieben) definiert welche Operationen angeboten werden.

Weitere Informationen zu diesem Thema können unter folgenden Links gefunden werden:

SOA - Wikipedia
Service Oriented Architecture - Microsoft

4.1.5.Aspect Oriented Programming

Ein heißes Thema und für viele sicherlich interessant. Und ein Blick über den Tellerrand schadet sowieso nie. Daher ein paar gute Links dazu:
Aspect oriented programming in C# :- Part I
Aspect oriented programming in C# :- Part II
Using AOP in C#
AOP (Aspect Oriented Programming) in C#
Aspect Oriented Programming using .NET - AOP in C#
AOP Support for C# (PDF)

4.2.Analyse

4.2.1.Eigenen Code analysieren

Wo gibt es Verbesserungsmöglichkeiten? Sind für die Performance relevante Codepassagen in meiner Anwendung? Oder befinden sich sogar potentielle Sicherheitslöcher im Programm?

Wer bereits mit FxCop gearbeitet hat, kennt auch die entsprechende Visual Studio 2005 Funktionalität. FxCop wurde integriert und steht somit allen Nutzern zur Verfügung. Dazu gibt es den Tab "Code Analysis" in den Projekteigenschaften. Dort können ebenfalls auch die einzusetzenden Regeln ausgewählt werden. Auch die Aktivierung der Code Analyse kann in diesem Fenster eingestellt werden. Die Ergebnisse einer Analyse werden im Error Output angezeigt.

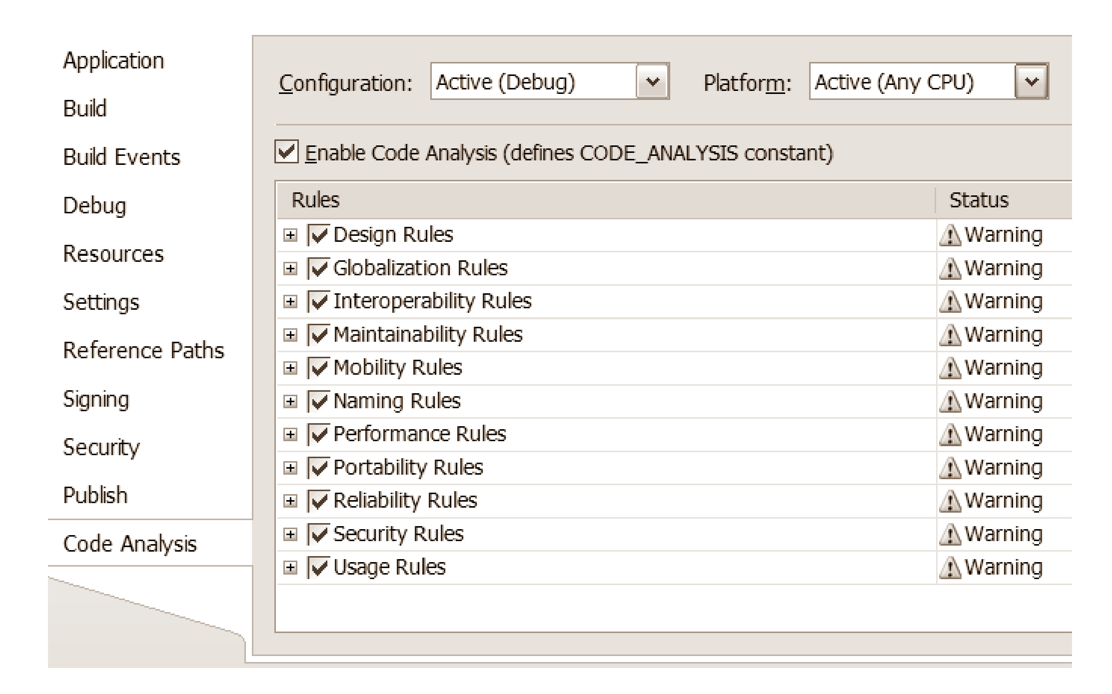
Generell empfiehlt es sich, Code Analysis zu verwenden, um potentielle Sicherheitsprobleme und weitere "kritische" Stellen zu finden. Allerdings sollten die Warnings an das jeweilige Anwendungsdesign "angepasst" werden, da nicht alle angewandt werden können.

4.2.2.Code Analysis als Hilfsmittel

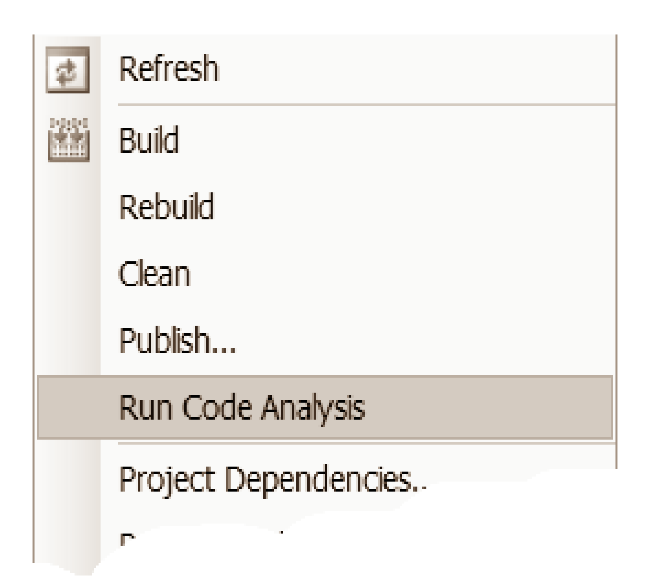
In meinem Beitrag Eigenen Code analysieren hatte ich bereits über **Code Analysis** berichtet. An dieser Stelle möchte ich diesen wichtigen Part noch einmal in Erinnerung rufen.

Visual Studio besitzt bereits eine integrierte Code Analyse und jede Menge Regeln, die beim Aufruf der Code Analysis aufgerufen werden. Von vielen als störend abgetan, besitzt die regelmäßige Analyse jedoch durchaus seine Berechtigung.

Die Einstellungen zur Analyse können in den Projekteigenschaften vorgenommen werden.



Wird der Compiler-Schalter CODE_ANALYSIS aktiviert, wird die Analyse bei jedem Kompilierungs-Vorgang ausgeführt. Wer dies möchte, kann die Analyse auch manuell starten. Hierzu ist das Kontext-Menü eines Projektes zu öffnen und **Run Code Analysis** aufzurufen.



In einigen Fällen kann es vorkommen, dass bei Durchsicht des Codes, dieser als korrekt definiert wird, obwohl durch die Code Analyse eine Warnung ausgegeben wurde. Für derartige Fälle kann diese mit Hilfe des `SuppressMessage`-Attributes (SuppressMessageAttribute) unterdrückt werden.

Ein Beispiel möchte ich an dieser Stelle nicht vorenthalten.

Nun stellt sich eventuell noch die Frage, wofür Code Analysis denn eigentlich gut sein soll und welchen Nutzen man davon hat.
Je nach definierten Regeln (idealerweise sollten alle Regeln aktiviert sein), werden Hinweise auf undokumentierte Stellen gegeben, unsichere Codestellen entdeckt, auf obsolete Klasse hingewiesen und weitere Hilfestellungen gegeben. Vor allem

Sourcecode-Kommentare sollte in gesundem Maße durchgeführt werden, auch wenn manche Statistiken behaupten, dass sich die meisten Fehler gerade in gut dokumentierten Stellen befinden bzw. "guter" Sourcecode keiner Dokumentation bedarf. Dennoch sollten derartige Hinweise ernst genommen und Warnungen aus der Analyse gering gehalten werden.

4.3.Software Testing

4.3.1.Das Übel Software-Testing

Gestern hatte ich ein längeres Gespräch mit einem befreundeten Entwickler. Basis der Diskussion war das Thema Software-Testing in allen Formen, Varianten und warum die Akzeptanz so gering ausfällt.

Fakt ist (zumindest laut meiner persönlichen Erfahrung), dass nur sehr wenige Firmen Software wirklich testen. Damit meine ich nun weniger, das rein zufällige Finden von offensichtlichen Bugs durch unkontrolliertes - an Zuckungen erinnerndes - Herumklicken auf der GUI, sondern vielmehr der Einsatz von einschlägigen Hilfsmitteln á Unit-Tests und Co.

Wenn ich nun kommuniziere, dass es am zusätzlichen Aufwand liegt, dann wiederhole ich mich, denn dieser dürfte es anscheinend nicht sein. Vielmehr daran, dass viele Entwickler nur dann Spaß am Programmieren haben, wenn Neues geschaffen werden kann. Beim Schreiben von Tests muss jedoch bereits entwickeltes geprüft werden (in vielen Fällen), was einer Wiederholung gleich kommt. Um es auf den Punkt zu bringen: es ist einfach langweilig. Daher wird dieser wichtige Schritt einfach nicht durchgeführt.

Das Resultat? Nun, wer würde sich ein Auto kaufen, welches jedes Mal auf der Autobahn abstirbt, weil der Motor aufgrund der andauernden Belastung überhitzt, also quasi einen Überlauf verursacht. Oder man stelle sich vor, dass das Aufblendlicht nicht funktioniert, weil der Kontakt zur Birne fehlt, was vergleichbar mit einer Null-Reference-Exception wäre. Viele viele Beispiele gäbe es an dieser Stelle. Grundsätzlich möchte jeder ein voll funktionstüchtiges Auto. Genauso verhält es sich mit Software. Wenn andere Branchen so fehlerbestückte Produkte ausliefern würden, wie es Software-Branche tut, könnten wir vermutlich mit keinem Gerät bzw. keinem Produkt tatsächlich etwas anfangen. Das stimmt sehr nachdenklich.

In diesem Sinne wünsche ich einen guten Start ins Wochenende und auf dass es in Zukunft nicht auch notwendig ist, die Software zum Service zu bringen oder gar jährlich ein Pickerl (TÜV) machen zu lassen.

4.3.2.Unit-Tests und Aufwand

Danke für die vielen Rückmeldungen, die ich vor allem in Form von Emails erhalten habe. Dabei stand fast immer die Frage nach dem Aufwand im Vordergrund.

Zu diesem Thema kann ich folgendes sagen/behaupten:

Natürlich verursacht das Erstellen von Unit-Tests einen entsprechenden Aufwand, da jede Methode in allen erdenklichen Varianten getestet werden soll. Sinnvollerweise wird der Unit-Tests parallel zur zu testenden Klasse entwickelt. Dadurch erhöht sich zwar der Aufwand für die Entwicklung von Klassen, man bedenke aber das große ABER:

Durch die ständigen Test-Durchläufe kristallisieren sich sehr schnell Fehler heraus bzw. wo es nach einer durchgeführten Änderung ein wenig zwickt. Sollte dann doch einmal ein Fehler durchrutschen, kann mit Hilfe der Unit-Tests dieser recht schnell nachvollzogen bzw. überhaupt gefunden werden. Das erspart sehr viel Zeit. Vermutlich mehr, als man sich durch das Nichtschreiben von Unit-Tests sparen würde.

Jeder Leser kann sich nun selbst ein Bild davon machen. Einfach die durchschnittliche Anzahl der Bugs heranziehen und kurz darüber nachdenken, wie schwer so manche davon zu finden ist. Viele davon wären mit Unit-Tests erst gar nicht zustande gekommen.

Ich hoffe dieser Beitrag regt zum Nachdenken und Nachrechnen an und liefert so für jeden von Euch ein Ergebnis bezgl. der Sinnhaftigkeit von Unit-Tests.

4.3.3. Grundlagen: Testgetriebene Entwicklung (test-driven Development)

Die testgetriebene Entwicklung kommt aus dem Bereich der agilen Softwareentwicklung und legt fest, dass Software-Tests vor der Entwicklung der zu testenden Komponenten erstellt werden. Zur Verwendung kommen so genannte **Grey-Box-Tests**. Diese vereinen die Vorteile der White-Box-Tests [1] und der Black-Box-Tests [2] in sich:

1. Die Software-Tests werden vom Entwickler der Komponente selbst geschrieben (White-Box-Test)
2. Der Test wird ohne Kenntnis der Komponente (da diese ja noch nicht entwickelt ist) erstellt (Black-Box-Test)

Der Vorteil dieser Variante liegt darin, dass nicht um Fehler "herumgetestet" wird. Das heißt die Tests werden nicht der Komponente angepasst, sondern die Komponente muss so funktionieren, wie dies im Test festgelegt wurde.

Bei dieser Art der Tests empfiehlt es sich jedoch eine hohe Disziplin an den Tag zu legen und zusätzliche Arbeitsweisen aus der agilen Softwareentwicklung zu nutzen. Zusätzlich sollte nicht auf Black-Box-Tests verzichtet werden.

[1] White Box Tests

[2] Black Box Tests

4.3.4.Grundlagen: White Box Tests

Bei den White Box Tests wird im Gegensatz zu den Black Box Tests die Implementierung einer Komponente getestet. Dies bedeutet, dass vor dem Entwickeln des Tests ein Blick auf den Sourcecode durchaus erlaubt ist.

Ziel der White Box Tests ist es, dass bezüglich des Sourcecodes bestimmte Hinlänglichkeitskriterien erfüllt werden. Eine komplette Fehlerfreiheit kann durch diese Tests nicht sichergestellt werden. Eine Kombination mit anderen Test-Methoden ist anzuraten.

Weitere Methoden der Softwaretests:

- Black Box Tests
- Grey Box Tests

4.3.5.Grundlagen: Black Box Tests

Die Black-Box-Tests gehören den allgemeinen Softwaretests an und stellen eine Methode dar, Tests für Komponenten zu entwickeln, die noch nicht erstellt wurden. Für die einzelnen Tests wird die Spezifikation der Methoden, Klassen etc. herangezogen, jedoch nicht die dahinter liegende Implementierung (funktionsorientiertes Testen).

Das Ziel besteht darin, die Software hinsichtlich der Spezifikationen zu prüfen. Diese geben Schnittstellen, Definitionen und Ergebnisse vor, welche von den einzelnen Komponenten einzuhalten sind und entsprechend geprüft werden müssen.

Ein Black-Box-Test kann jedoch sehr aufwändig sein und birgt auch ein weiteres Problem in sich:

In der Softwareentwicklung werden Komponenten ständig weiter entwickelt. Daraus ergeben sich neue Funktionalitäten, die ebenfalls getestet werden müssen. Diese Tests sind jedoch in den Black-Box-Tests nicht enthalten, da diese bereits vor der Komponente entwickelt werden. Erfolgreiche Testläufe bedeuten daher jedoch nicht zwangsweise, dass die gesamte Funktionalität der Komponente zu einem späteren Zeitpunkt getestet wurde, sondern lediglich die ursprüngliche Spezifikation.

Zusätzlich gibt es noch weitere Verfahren:

- Grey Box Test
- White Box Test

4.3.6.Unit-Tests mit Visual Studio .NET

Einführung und Ausblick

Bei fast allen Fragestellungen in diversen Foren gibt es Fragen zu Fehlern, die auf dreierlei Arten auftreten:

1. Mangelndes Studieren der zur Verfügung stehenden Quellen. Dies inkludiert nicht nur diverse Artikel, die es im Internet zu finden gibt (ein Beispiel hier wäre das MSDN), sondern auch in Form von Büchern oder den zu den verwendeten Werkzeugen mitgelieferten Ressourcen.

2. Fehlendes Verständnis und Wissen rund um den Debugger. Immer wieder muss ich feststellen, dass der Debugger wenig bis gar keine Beachtung findet, obwohl sich durch ihn das wohl beste "Werkzeug" überhaupt offenbart. Auf schnelle Art und Weise können nicht nur Syntax-Fehler gefunden werden, nein, es können auch Variableninhalte abgefragt werden, Objektinformationen bezogen und sogar Abfragekommandos an den Compiler übergeben werden.

3. Ignoranz von Unit-Tests. Die Verwendung von Unit-Tests wird in vielen Fällen nur Profis zugetraut und aus diesem Grunde wird ihnen kaum Beachtung geschenkt - oder aber, einfach das fehlende Wissen um diese Möglichkeit.

Dieser Artikel soll zeigen, dass selbst Programmier-Anfänger sehr einfach Unit-Tests verwenden können.

Begriffserklärung

Bevor wir aber tiefer in die Materie einsteigen, muss der Begriff Unit-Test erklärt werden.

Durch Unit-Tests ist es möglich, Testklassen zu schaffen, die automatisiert andere, vorhandene Klassen testen.

So werden pro Testklasse folgende Dinge der zu testenden Klasse angegeben und getestet:

- alle Methoden
- alle Überladungen
- in allen erdenklichen Übergabeparametern (auch sinnlosen Varianten)

Vor allem die "sinnlosen Varianten" sind für die Tests sehr wertvoll: Dadurch werden Überläufe, und vor allem auch Fehleingaben durch User (diese passieren immer und überall) mit in den Test einbezogen und im Fehlerfalle kann dies bevor das Produkt den User erreicht nachgebessert werden.

Der Sinn hinter solchen Tests ist der, dass wie oben beschrieben, problematische Parameterübergaben im Fehlerfalle behandelt werden (da auf das Testergebnis entsprechend zu reagieren ist). Da eine Software laufend weiterentwickelt wird, werden auch ständig an allen erdenklichen Stellen Änderungen vorgenommen. Dadurch ist es in manchen Fällen schwer zu sagen, ob eine andere Stelle noch korrekt funktioniert. Durch diese Unit-Tests kann dies einfach festgestellt werden. Dazu sind nach der durchgeführten Änderung die Tests auszuführen und wenige

Sekunden bis Minuten später (Abhängig von der Projektgröße) hat der Programmierer Gewissheit.

Notwendige Tools

Zu Beginn muss natürlich geklärt werden, welche Tools notwendig sind, um Unit-Tests unter dem Visual Studio verwenden zu können.

Aus meiner Erfahrung haben sich folgende Produkte als äußerst hilfreich erwiesen:

- TestDriven.NET [1]
- NUnit [2]

Bei TestDriven.NET handelt es sich um Unit-Testing Add-In für Visual Studio, welches mit unterschiedlichsten Unit-Testing-Tools zusammenarbeiten kann. Zu erwähnen wären hier NUnit, MbUnit und csUnit.

Bei NUnit handelt es sich um ein Test-Unit Framework, mit dessen Hilfe Unit-Tests durchgeführt werden können.

Installation

An diesem Punkt möchte ich nur an die entsprechenden Installations-Hinweise der Hersteller verweisen:

NUnit: http://www.nunit.org/index.php?p=installation&r=2.2.6

Bei TestDriven.NET ist lediglich die Installationsdatei zu starten. Daraufhin wird das Add-In im Visual Studio registriert und steht fortan zur Verfügung.

Vorabeiten

Um nun für ein neues oder bereits bestehendes Projekt Unit-Tests anzuwenden, sind kleine Vorabeiten notwendig.

Idalerweise empfiehlt es sich, Unit-Tests in ein eigenes Projekt auszulagern. Dieses Projekt muss natürlich Teil der Visual Studio Solution sein.

Nach dem Anlegen dieses Projektes, ist eine Referenz auf die nunit.framework.dll zu setzen. Ist dies geschehen, kann es mit einem konkreten Beispiel weitergehen.

Ein konkretes Beispiel

Gehen wir davon aus, dass unser Projekt einen Logger besitzt. Dieser hat die Aufgabe, allfällige Fehler in eine Logdatei zu schreiben. Natürlich muss dieser getestet werden, ob er auch den an ihn gestellten Anforderungen gerecht wird.

Dazu erstellen wir eine TestKlasse `LoggerTest`. Die neue Klasse muss zusätzlich mit dem Attribut `[TestFixture]` markiert werden. Danach erstellen wir die

einzelnen Methoden, welche die einzelnen Tests darstellen. Beispielsweise könnte dies folgendermaßen aussehen:

```
public void LoggerLog()
{
    try
    {
        Logger log = new Logger();
        log.LogPath = @"C:temptemp.log";
        log.Log("test");
        Assert.IsTrue(true);
    }
    catch (Exception ex)
    {
        Assert.Fail(ex.Message);
    }
}
```

Wie an diesem Beispiel zu sehen ist, müssen die zu testenden Methoden mit dem Attribut `[Test]` versehen werden.

Mit einem Rechtsklick auf die entsprechende Datei kann nun mittels "Run Test(s)" der Test gestartet werden. Wird dieser Befehl auf das Projekt angewandt, werden alle darin enthaltenen Tests gestartet.

Nach dem Durchlauf der Tests erscheint im Ausgabe-Fenster die Angabe, welche Tests durchgelaufen (also erfolgreich waren) und welche nicht.

Wann ein Test als erfolgreich und wann als nicht erfolgreich gilt, muss vom Entwickler selbst angegeben werden. Im Falle einer Exception darf der Test natürlich nicht als erfolgreich angeführt sein. Wurden alle Punkte durchlaufen und stimmt das Ergebnis, kann der Test als erfolgreich markiert werden.

Dafür zuständig ist die Klasse `Assert`. Diese hat einige Methoden, die für diese Aufgabe sehr hilfreich sind:

`Assert.Fail`: Hier ist ein String zu übergeben, der angibt, um welchen Fehler es sich handelt. Dies kann die `Message`-Eigenschaft einer Exception sein, oder auch ein selbst definierter String, um mitzuteilen wo bei was ein Fehler aufgetreten ist.

`Assert.IsTrue`: Hier werden normalerweise Vergleiche angegeben. Beispielsweise das Ergebnis der Methode und das Ergebnis, welches die Methode liefern sollte. Sind beide Ergebnisse ident, ist der Test durchgelaufen, andernfalls nicht.

`Assert.IsFalse`: Dies funktioniert wie `Assert.IsTrue`, nur in die entgegengesetzte Richtung.

Konklusio

Dieses kleine Tutorial lieferte einen kurzen Einblick in das Thema Unit-Tests unter Visual Studio und zeigt durchaus, dass sich dahinter keine komplizierten Abläufe

verbergen. Stöbert man ein wenig in den angegebenen Internet-Ressourcen herum, können sehr schnell gute und hilfreiche Ergebnisse erreicht werden.

Referenzen

[1] http://www.testdriven.net
[2] http://www.nunit.org

4.3.7.Nicht ausgeführte UnitTests mit TestDriven.NET

Grade wieder drüber gestolpert:

Mit TestDriven.NET ist es möglich UnitTests bequem aus Visual Studio zu starten und auszuwerten. Über das Kontextmenü der Projektmappe ist es außerdem möglich, alle vorhandenen Tests des Projekts zu starten. Allerdings sollte man darauf achten, dass die Test-Klassen auch tunlichst als `public` gekennzeichnet sind. Ansonsten werden diese Tests nicht ausgeführt.

Wer noch nie etwas von UnitTests oder TestDriven.NET gehört hat, sollte einen Blick in die aktuelle Ausgabe der Zeitschrift *Visual Studio one* werfen und sich den Artikel *Visual Studio goes Unit Testing* von Norbert Eder zu Gemüte führen.

4.3.8.Externes Configuration File benutzen

Heute musste ich in einer Konsolen-Anwendung ein Configuration File benutzen, dass nicht direkt zur Anwendung gemappt ist. Noch in blasser Erinnerung an das 1.1 Framework hatte mich mich schon mit dem Gedanken abgefunden, mir einen XML-Parser zu schreiben, der die entsprechend Einträge sucht. Tatsächlich aber bietet das Framework 2.0 dafür entsprechende Klassen und Methoden. Somit ist das Problem mit ein paar Zeilen schnell gelöst:

```
ExeConfigurationFileMap configFile = new ExeConfigurationFileMap();
configFile.ExeConfigFilename = "test.config";

Configuration config = ConfigurationManager.OpenMappedExeConfiguration(configFile,
ConfigurationUserLevel.None);
AppSettingsSection section = (AppSettingsSection)config.GetSection("appSettings");

string configValue = section.Settings["Folder"].Value;
```

4.3.9.ToolTips einzelner Items einer CheckBoxList setzen

In einem Forum tauchte die Frage auf, wie man die Eigenschaft `ToolTip` einzelner CheckBoxen einer CheckBoxList setzt. Das CheckBoxList Control bietet selbst eine Eigenschaft `ToolTip`, allerdings setzt man dann gleich den `ToolTip` für alle CheckBoxen.

Die Eigenschaft `ToolTip` selbst bewirkt, dass das HTML-Attribut `Title` gesetzt wird. Man muss also nur durch alle CheckBoxen iterieren und dieses Attribut manuell setzen. Der Code hierfür ist recht einfach:

```
for (int i = 0; i < CheckBoxList1.Items.Count; i++)
{
   ListItem item = (ListItem)CheckBoxList1.Items[i];
   item.Attributes.Add("title", "TestTooltip " + i);
}
```

Um den ToopTip einzelner Items der Liste direkt setzen, ohne durch die gesamte Liste zu iterieren, habe ich mir eine kleine Methode geschrieben:

```
private void SetCheckBoxToolTipByValue(string value, string toolTip)
{
  ListItem item = (ListItem)CheckBoxList1.Items.FindByValue(value);

  if (item != null)
    item.Attributes.Add("title", toolTip);
}
```

Die Methode sucht anhand eines `Values` den ensprechenden Eintrag und setzt den `Title`.

Das komplette Beispielprojekt gibt es hier..

4.4.Deployment

4.4.1.Deploying unter .NET 3

Alles zum Thema Deployment unter .NET 3 findet sich wohl unter [1]. Wer also schon früh informiert sein will, der sollte sich diesen Link genauer ansehen.

[1] Deploying Microsoft .NET Framework Version 3

4.4.2..NET ClickOnce via Firefox

.NET ClickOnce ist mit Firefox nicht durchführbar und blieb daher den Internet Explorer Benutzern vorenthalten. Bei den Firefox Add-Ons befindet sich allerdings eines namens **FFClickOnce** [1] welches diese Funktionalität hinzufügt.

Firefox: 1.5 – 2.0.0.*

Wer sich über das Thema ClickOnce informieren möchte, der sei auf den Artikel ClickOnce des MSDN's verwiesen.

[1] FFClickOnce Download

4.5.Design Patterns

4.5.1.Einführung

Bevor unterschiedlichste Design-Patterns erklärt werden, muss definiert werden, worum es sich bei Design-Patterns handelt.

Hier eine kurze Definition:

"Jedes Muster beschreibt in unserer Umwelt ein beständig wiederkehrendes Problem und erläutert den Kern der Lösung für dieses Problem, so dass diese Lösung beliebig oft angewendet werden kann, ohne sie jemals ein zweites Mal gleich auszuführen"

A Pattern Language von Christopher Alexander, Oxford University Press 1977.

Was genau bedeutet dies? Jedes Pattern wird/wurde für eine bestimmte Aufgabe entwickelt und kann darin generisch eingesetzt werden. Sie sollen also häufig gestellte Aufgaben vereinfachen. Weiters handelt es sich um eine Beschreibung der zusammen arbeitenden Objekte und Klassen.

Zur weiterführenden Recherche möchte ich erwähnen, dass es unterschiedliche Typen von Design-Patterns gibt:

- Erzeugungs-Pattern
- Struktur-Pattern
- Verhaltens-Pattern

Sollten nähere Erklärungen von Nöten sein, werde ich dies an den entsprechenden Stellen nachholen, oder zu den unterschiedlichen Typen noch entsprechende Einträge schreiben.

4.5.2.Geschichte

Unter Was sind Design Patterns wurde der Begriff **Design Pattern** bereits definiert und auch die unterschiedlichen Arten wurden angeführt.

In diesem Beitrag wird auf die Geschichte der Design Patterns eingegangen.

Bereits in den 1970er Jahren wurde die erste Sammlung von Entwurfsmustern erstellt - allerdings von einem Architekten namens Christopher Alexander. Die Idee dahinter hat sich seitdem nicht verändert. Nur fand seine Sammlung wenig Anklang

unter anderen Architekten, in der Softwareentwicklung wurde die Idee jedoch bald darauf aufgegriffen und erfreut sich großer Beliebtheit. Ende der 1980er wurde die Sammlung von Christopher Alexander von Kent Beck und Ward Cunningham aufgegriffen und entwickelten auf deren Basis Entwurfsmuster für grafische Benutzerschnittstellen.

Eine neue Ära begann dann mit Erich Gamma. Nach seiner Promotion an der Universität Zürich, 1991, ging er in die USA und verfasste zusammen mit Richard Helm, Ralph Johnson und John Vlissides das Buch *Design Patterns - Elements of Reusable Object-Oriented Software*. In diesem Buch wurden 23 Design Patterns beschrieben. Dies verhalf den Entwurfsmustern zum Durchbruch. Die vier Autoren sind gemeinhin auch unter **Gang of Four (GoF)** bekannt.

Zur Übersichtlichkeit verwendete die GoF ein einheitliches Schema um die Design Patterns zu beschreiben. Nachfolgend eine kurze Übersicht:

- Mustername und Klassifikation
- Zweck
- Synonyme
- Motivation
- Anwendbarkeit
- Struktur
- Beteiligte Klassen (Akteure)
- Zusammenspiel der involvierten Klassen
- Vor- und Nachteile
- Implementierung
- Beispielcode
- Praxiseinsatz
- Querverweise

Anhand dieses Schemas konnte ausreichend Information zum entsprechenden Design Pattern geliefert werden (Wann ist es einsetzbar, etc.).

4.5.3.Command Pattern

Das nächste Pattern, welches ich in meiner Rubrik Patterns vorstellen möchte, ist das Command-Pattern.

Defintion

Das Command Pattern ermöglicht die Repräsentation von Aktivitäten in eigenständigen Objekten.

Was bedeutet dies in der Praxis?

Innerhalb der selben Struktur können unterschiedliche Commands ausgeführt werden, die jeweils eine bestimmte Aufgabe besitzen. Unabhängig davon, welche Aufgabe ein Command hat, muss das ausführende Konstrukt nicht verändert werden. Zusätzlich besteht die Möglichkeit, Commands in beispielsweise einer Queue abzulegen um hintereinander abgearbeitet zu werden.

Beispiel-Implementierung

Der Aufbau des Command Patterns wird zwar in manchen Fällen recht aufwändig und kompliziert dargestellt, gestaltet sich in der Praxis jedoch als sehr einfach. Es wird ein Interface `ICommand` erstellt welches die notwendige Methode `Execute()` zur Verfügung stellt. Jeder konkrete Command implementiert dieses Interface und die Methode `Execute()` wird dabei mit der Logik des Commands befüllt. In einigen Fällen macht es auch durchaus Sinn, einen abstrakten Command einzufügen, der bereits Basisfunktionalitäten zur Verfügung stellt.

```
public interface ICommand
{
    string Message { get;set;}
    void Execute();
}
```

Dieses Interface schreibt also die Methode `Execute()` vor, die von jedem Command implementiert werden muss. Weiters wird eine Eigenschaft `Message` vorgeschrieben, die eine entsprechende Nachricht speichern soll. Konkreter Anwendungsfall wäre eine Fehlermeldung wenn der Command auf einem entfernten Host ausgeführt wird (kann als eine Ergänzung zu einem State gute Dienste leisten).

In diesem Beispiel wird nun ein abstrakter Command erstellt, der die Eigenschaft als Basisfunktionalität zur Verfügung stellt:

```
public abstract class Command : ICommand
{
    private string message = null;

    public string Message
    {
        get { return this.message; }
        set { this.message = value; }
    }

    public abstract void Execute();
}
```

Alle Commands, die nun vom abstrakten Command ableiten, müssen die Eigenschaft `Message` nicht mehr implementieren, da diese bei der Ableitung übernommen wird. Lediglich die Methode `Execute` ist zu überschreiben. Dies wird anhand des folgenden Sourcecodes gezeigt:

```
public class ConcreteCommand : Command
{
    #region ICommand Members

    public override void Execute()
    {
        this.Message = "This is a concrete command implementation.";
    }

    #endregion
}
```

Hier passiert nichts anderes, als dass die Eigenschaft `Message` mit einem String befüllt wird.

Weiters wird von mir noch eine Klasse Executor benutzt, der für die Ausführung der Commands zuständig ist.

```
public class Executor
{
    public void ExecuteCommand(ICommand ic)
    {
        ic.Execute();
    }
}
```

Wie hier zu sehen ist, wird ein ICommand übergeben, der anschließend ausgeführt wird. Dadurch kann jeder beliebige Command ausgeführt werden und eine Erweiterung ist nur dann notwendig, wenn der Executor selbst diverse neue Features bekommt.

Zusammenfassung

Da im Falle des Command Patterns nur mehr die einzelnen Commands implementiert werden müssen, das ausführende Framework sich jedoch nicht ändert, eignet sich dieses Pattern sehr gut für eine Client-Server Kommunikation und wird daher auch oft eingesetzt, wenn Clients beispielsweise mit Web-Services interagieren.

Nachfolgend findet sich ein kleines Testprogramm, welches den gesamten Ablauf dieses Patterns zeigt.

Beispiel-Anwendung: Download

4.5.4.Singleton

Das Singleton-Pattern ist wohl eines der bekanntesten Design-Pattern überhaupt. Ziel ist es, dass es innerhalb einer Anwendung von einer Klasse nur ein Objekt gibt. Daher kann global auf dieses Objekt zugegriffen werden.

Einsatzgebiete für das Singleton-Pattern finden sich einige:

- Logging-System welches Logging-Daten in eine Datei schreibt
- Druckerqueue in der alle Dokumente der Reihe nach abgearbeitet werden sollen
- Implementierung eines globalen Storages
- etc.

Das Singleton-Pattern gehört zu den Erzeugungsmustern (engl. Creational Patterns).

Im nachfolgenden Sourcecode-Beispiel wird ein thread-sicherer Singleton abgebildet. Thread-sicher bedeutet, immer nur ein Thread auf eine Ressource zugreifen kann. Dadurch können ungewollte Effekte vermieden werden.

```
public class Singleton
{
    private static Singleton instance = null;
```

```
    private static object lockObject = new object();

    private Singleton()
    {

    }

    public static Singleton GetInstance()
    {
        lock (lockObject)
        {
            if (instance == null)
                instance = new Singleton();
        }
        return instance;
    }
}
```

Zu beachten ist, dass ein Singleton einen privaten Konstruktor hat. Dadurch wird gewährleistet, dass außerhalb der Klasse kein Objekt der Singleton-Klasse erstellt werden kann. Dafür wird die Methode `GetInstance()` verwendet. Hier findet die Überprüfung statt, ob es bereits eine Instanz gibt. Wenn nicht, wird eine neue erstellt, ansonsten wird die bestehende Instanz zurückgegeben. Prinzipiell kann die Methode `GetInstance()` auch als Getter-Property abgebildet werden.

Die Thread-Sicherheit bringt uns in diesem Fall das Schlüsselwort `lock`. Diesem kann als Parameter ein Objekt - welches gesperrt werden soll - übergeben werden. Solange das übergebene Objekt durch einen Lock gesperrt wird, kann kein anderer Thread darauf zugreifen.

Weitere Informationen zu diesem Thema sind unter anderem im WeWeblog von Dirk Primbs zu finden.

4.5.5.Proxy-Pattern

Ein Proxy kann an vielen Stellen eingesetzt werden. Grundlegend handelt es sich dabei um einen Platzhalter für das tatsächlich aufzurufende Objekt. Das heißt, es leitet vom gleichen Interface ab, besitzt die gleichen Methoden, leitet aber alle Anfragen an das dahinter liegende Objekt weiter. Aber welchen Vorteil besitzt dieses Pattern nun?

Durch dieses Pattern ist es möglich, Änderungen und Prüfungen einzuführen, ohne das eigentliche Objekt abändern zu müssen. So können beispielsweise Sicherheitsprüfungen stattfinden (**Protection Proxy**). Dies bedeutet, dass das Proxy-Objekt zuerst überprüft, ob der Aufrufer über die notwendigen Rechte verfügt, bevor der Aufruf weitergeleitet und abgearbeitet wird. Weiters wird häufig ein **Remote Proxy** verwendet, der die Anfrage kodiert, an das echte Objekt sendet, die Antwort dekodiert und zurück gibt. Dieser Vorgang ist unter anderem dann interessant, wenn Application Domains erstellt werden sollen, die zur Laufzeit die Möglichkeit bieten sollen, entladen werden zu können. Schließlich wäre da noch **Cache Proxies**, die bestimmte Daten des eigentlichen Objektes cachen und so diverse Vorgänge beschleunigen können. Das sind jedoch nicht die einzigen Möglichkeiten. So gibt es noch **Synchronization Proxies** und viele weitere.

Ein einfaches Grundgerüst eines Proxies findet sich nachfolgend:

```
using System;
using System.Collections.Generic;
using System.Text;

namespace ProxyPattern
{
    class Program
    {
        static void Main(string[] args)
        {
            CalculatorProxy proxy = new CalculatorProxy();
            Console.WriteLine(
                String.Format("12 + 17 = {0}", proxy.Add(12, 17))
                );
            Console.WriteLine(
                String.Format("12 - 17 = {0}", proxy.Sub(12, 17))
                );

            Console.Read();
        }
    }

    public interface ICalculator
    {
        decimal Add(decimal x, decimal y);
        decimal Sub(decimal x, decimal y);
    }

    public class Calculator : ICalculator
    {
        public decimal Add(decimal x, decimal y)
        {
            return x + y;
        }

        public decimal Sub(decimal x, decimal y)
        {
            return x - y;
        }
    }

    public class CalculatorProxy : ICalculator
    {
        private Calculator _calculator = new Calculator();

        public decimal Add(decimal x, decimal y)
        {
            return _calculator.Add(x, y);
        }

        public decimal Sub(decimal x, decimal y)
        {
            return _calculator.Sub(x, y);
        }
    }
}
```

Dies stellt die einfachste Variante eines Proxies dar. Ein vereinfachter Protection Proxy könnte beispielsweise so aussehen (Erweiterung des obigen Beispiels):

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Security;

namespace ProxyPattern
{
```

```
    class Program
    {
        static void Main(string[] args)
        {
            try
            {
                CalculatorProxy proxy = new CalculatorProxy("test2");
                Console.WriteLine(
                    String.Format("12 + 17 = {0}", proxy.Add(12, 17))
                    );
                Console.WriteLine(
                    String.Format("12 - 17 = {0}", proxy.Sub(12, 17))
                    );

            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
            Console.Read();
        }
    }

    public interface ICalculator
    {
        decimal Add(decimal x, decimal y);
        decimal Sub(decimal x, decimal y);
    }

    public class Calculator : ICalculator
    {
        public decimal Add(decimal x, decimal y)
        {
            return x + y;
        }

        public decimal Sub(decimal x, decimal y)
        {
            return x - y;
        }
    }

    public class CalculatorProxy : ICalculator
    {
        private Calculator _calculator = new Calculator();
        private string _password = "test";
        private string _givenPassword = null;

        public CalculatorProxy(string password)
        {
            this._givenPassword = password;
        }

        public decimal Add(decimal x, decimal y)
        {
            if (this._givenPassword == this._password)
                return _calculator.Add(x, y);
            else
                throw new SecurityException("Not allowed: Add");
        }

        public decimal Sub(decimal x, decimal y)
        {
            if (this._givenPassword == this._password)
                return _calculator.Sub(x, y);
            else
                throw new SecurityException("Not allowed: Sub");
        }
    }
}
```

Ich möchte nur darauf hinweisen, dass ein Protection Proxy entsprechend abgesichert sein sollte. Die obige Variante dient lediglich der Veranschaulichung und sollte so nicht verwendet werden.

4.5.6.Builder Pattern in C#

Eine Diskussion mit Thomas veranlasste mich, einige Beispiele für ein Builder Pattern im .NET Framework zu suchen, zu beschreiben und selbst ein kleines Beispiel aus der Praxis zu liefern.

Einführung

Grundlegend zählt das Builder Pattern zu den **Object Creational Patterns** - es werden durch dieses Entwurfmuster also Objekte erstellt. Sinn und Zweck des Builder Patterns ist es, den Objekt-Erstellungsprozess von der Repräsentation des Objektes zu entkoppeln. Dadurch ist es möglich, mit demselben Erstellungsprozess unterschiedliche Repräsentationen erstellen zu können.

Beispiel aus dem .NET Framework

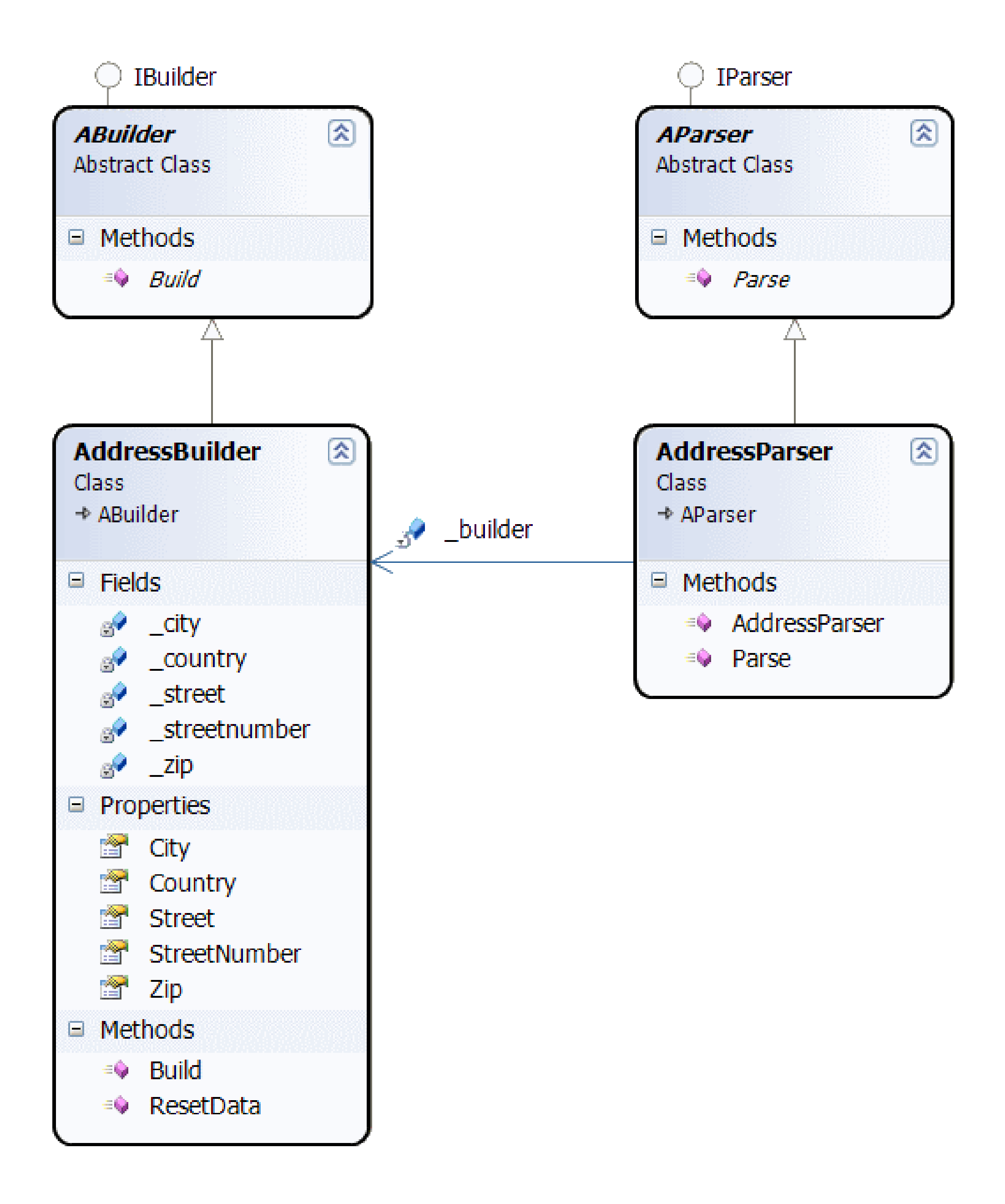
Ein sehr gutes Beispiel aus dem .NET Framework ist durch den `DbConnectionStringBuilder` geben. Dieser stellt die Basisklasse für stark typisierte Ableitungen dar (`SqlConnectionStringBuilder` etc.). Sehen wir uns an dieser Stelle einen kurzen Sourcecode an:

```
SqlConnectionStringBuilder connStringBuilder =
  new SqlConnectionStringBuilder();
connStringBuilder.DataSource = "(local)";
connStringBuilder.InitialCatalog = "MyDatabase";
connStringBuilder.UserID = "username";
connStringBuilder.Password = "$password%";
SqlConnection sqlConn =
  new SqlConnection(connStringBuilder.ToString());
```

Mit Hilfe des `SqlConnectionStringBuilder` ist es möglich, einfach einen syntaktisch korrekten `ConnectionString` zu erstellen, woraus in weiterer Folge ein `Connection`-Objekt gebildet werden kann.

Ein eigenes Beispiel

Nun sehen wir uns anhand eines eigenen Beispiels die Implementierung des Builder Patterns an. Grundlage bilden Adressdaten, die in einer XML-Datei vorliegen und importiert werden müssen. Den Part des Importierens ersparen wir uns. Die erstellten Objekte werden lediglich in eine Liste gelegt und anschließend per **ToString** ausgegeben.



Wie in der Abbildung zu sehen ist, besteht diese Implementierung des Builder-Patterns aus zwei wichtigen Teilen:

- AddressBuilder
- AddressParser

Der Parser selbst ist kein echter Bestandteil des ursprünglichen Builder-Patterns, wurde jedoch in diesem Beispiel zum Parsen der Adressdaten herangezogen und in das Pattern integriert, um die Zuständigkeiten der Objekte sauber zu trennen.

```
public class AddressParser : AParser
{
    AddressBuilder _builder = null;
    public AddressParser(IBuilder builder)
    {
        _builder = (AddressBuilder)builder;
    }
```

```
    public override void Parse(XmlNode data)
    {
        if (data != null && _builder != null)
        {
            _builder.ResetData();
            XmlNode xnStreet = data.SelectSingleNode("street");
            XmlNode xnStreetNumber =
                data.SelectSingleNode("streetnumber");
            XmlNode xnZip = data.SelectSingleNode("zip");
            XmlNode xnCity = data.SelectSingleNode("city");
            XmlNode xnCountry = data.SelectSingleNode("country");
            if (xnStreet != null)
                _builder.Street = xnStreet.InnerText;
            if (xnStreetNumber != null)
                _builder.StreetNumber = xnStreetNumber.InnerText;
            int zip = 0;
            if (xnZip != null)
                Int32.TryParse(xnZip.InnerText, out zip);
            _builder.Zip = zip;
            if (xnCity != null)
                _builder.City = xnCity.InnerText;
            if (xnCountry != null)
                _builder.Country = xnCountry.InnerText;
        }
    }
}
```

Der AddressBuilder selbst besitzt nun die Aufgabe, das eigentliche Address-Objekt zu erstellen und zurück zu liefern.

```
public class AddressBuilder : ABuilder
{
    private string _street = null;
    private string _streetnumber = null;
    private int _zip = 0;
    private string _city = null;
    private string _country = null;
    public string Street
    {
        get { return this._street; }
        set { this._street = value; }
    }
    public string StreetNumber
    {
        get { return this._streetnumber; }
        set { this._streetnumber = value; }
    }
    public int Zip
    {
        get { return this._zip; }
        set { this._zip = value; }
    }
    public string City
    {
        get { return this._city; }
        set { this._city = value; }
    }
    public string Country
    {
        get { return this._country; }
        set { this._country = value; }
    }
    public void ResetData()
    {
        this._city = null;
        this._country = null;
        this._street = null;
        this._streetnumber = null;
        this._zip = 0;
```

```
    }
    public override BaseObject Build()
    {
        Address address = new Address();
        address.City = this._city;
        address.Country = this._country;
        address.Street = this._street;
        address.StreetNumber = this._streetnumber;
        address.Zip = this._zip;
        return address;
    }
}
```

Die Ausführung der gesamten Anwendung erfolgt durch nachfolgenden Sourcecode:

```
List<Address> addresses = new List<Address>();
XmlDocument doc = new XmlDocument();
doc.Load("addresslist.xml");
XmlNodeList xnlAddresses = doc.SelectNodes("/addresses/address");
if (xnlAddresses != null && xnlAddresses.Count > 0)
{
    AddressBuilder builder = new AddressBuilder();
    AddressParser parser = new AddressParser(builder);
    foreach (XmlNode xnAddress in xnlAddresses)
    {
        parser.Parse(xnAddress);
        Address address = (Address)builder.Build();
        addresses.Add(address);
    }
}
foreach (Address a in addresses)
{
    Console.WriteLine(a.ToString());
    Console.WriteLine("---------------------");
}
Console.Read();
```

Fazit

Über dieses Pattern können durch kleine Anpassungen unterschiedlichste Typen abgewickelt werden, was sich sehr schnell positiv auswirkt und zudem alle für die passende Aufgabe notwendigen Schritte trennt. Dadurch lassen sich diese entsprechend anpassen bzw. gänzlich ersetzen.

Download Builder Pattern Demo

4.5.7.Proxy-Pattern: Beschreibung und Beispiele

Ein Proxy kann an vielen Stellen eingesetzt werden. Grundlegend handelt es sich dabei um einen Platzhalter für das tatsächlich aufzurufende Objekt. Das heißt, es leitet vom gleichen Interface ab, besitzt die gleichen Methoden, leitet aber alle Anfragen an das dahinterliegende Objekt weiter. Aber welchen Vorteil besitzt dieses Pattern nun?

Durch dieses Pattern ist es möglich, Änderungen und Prüfungen einzuführen, ohne

das eigentliche Objekt abändern zu müssen. So können beispielsweise Sicherheitsprüfungen stattfinden (**Protection Proxy**). Dies bedeutet, dass das Proxy-Objekt zuerst überprüft, ob der Aufrufer über die notwendigen Rechte verfügt, bevor der Aufruf weitergeleitet und abgearbeitet wird. Weiters wird häufig ein **Remote Proxy** verwendet, der die Anfrage kodiert, an das echte Objekt sendet, die Antwort dekodiert und zurück gibt. Dieser Vorgang ist unter anderem dann interessant, wenn Application Domains erstellt werden sollen, die zur Laufzeit die Möglichkeit bieten sollen, entladen werden zu können. Schließlich wäre da noch **Cache Proxies**, die bestimmte Daten des eigentlichen Objektes cachen und so diverse Vorgänge beschleunigen können. Das sind jedoch nicht die einzigen Möglichkeiten. So gibt es noch **Synchronization Proxies** und viele weitere.

Ein einfaches Grundgerüst eines Proxies findet sich nachfolgend:

```
using System;
using System.Collections.Generic;
using System.Text;
namespace ProxyPattern
{
    class Program
    {
        static void Main(string[] args)
        {
            CalculatorProxy proxy = new CalculatorProxy();
            Console.WriteLine(
                String.Format("12 + 17 = {0}", proxy.Add(12, 17))
                );
            Console.WriteLine(
                String.Format("12 - 17 = {0}", proxy.Sub(12, 17))
                );
            Console.Read();
        }
    }
    public interface ICalculator
    {
        decimal Add(decimal x, decimal y);
        decimal Sub(decimal x, decimal y);
    }
    public class Calculator : ICalculator
    {
        public decimal Add(decimal x, decimal y)
        {
            return x + y;
        }
        public decimal Sub(decimal x, decimal y)
        {
            return x - y;
        }
    }
    public class CalculatorProxy : ICalculator
    {
        private Calculator _calculator = new Calculator();
        public decimal Add(decimal x, decimal y)
        {
            return _calculator.Add(x, y);
        }
        public decimal Sub(decimal x, decimal y)
        {
            return _calculator.Sub(x, y);
        }
    }
}
```

Dies stellt die einfachste Variante eines Proxies dar. Ein vereinfachter Protection Proxy könnte beispielsweise so aussehen (Erweiterung des obigen Beispiels):

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Security;
namespace ProxyPattern
{
    class Program
    {
        static void Main(string[] args)
        {
            try
            {
                CalculatorProxy proxy = new CalculatorProxy("test2");
                Console.WriteLine(
                    String.Format("12 + 17 = {0}", proxy.Add(12, 17))
                    );
                Console.WriteLine(
                    String.Format("12 - 17 = {0}", proxy.Sub(12, 17))
                    );
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
            Console.Read();
        }
    }
    public interface ICalculator
    {
        decimal Add(decimal x, decimal y);
        decimal Sub(decimal x, decimal y);
    }
    public class Calculator : ICalculator
    {
        public decimal Add(decimal x, decimal y)
        {
            return x + y;
        }
        public decimal Sub(decimal x, decimal y)
        {
            return x - y;
        }
    }
    public class CalculatorProxy : ICalculator
    {
        private Calculator _calculator = new Calculator();
        private string _password = "test";
        private string _givenPassword = null;
        public CalculatorProxy(string password)
        {
            this._givenPassword = password;
        }
        public decimal Add(decimal x, decimal y)
        {
            if (this._givenPassword == this._password)
                return _calculator.Add(x, y);
            else
                throw new SecurityException("Not allowed: Add");
        }
        public decimal Sub(decimal x, decimal y)
        {
            if (this._givenPassword == this._password)
                return _calculator.Sub(x, y);
            else
```

```
                throw new SecurityException("Not allowed: Sub");
            }
        }
    }
```

Ich möchte nur darauf hinweisen, dass ein Protection Proxy entsprechend abgesichert sein sollte. Die obige Variante dient lediglich der Veranschaulichung und sollte so nicht verwendet werden.

4.6.Qualitäts-Management

4.6.1.Was ist Qualität?

Diese Frage ist wohl die essentiellste aller Fragen aus dem Qualitätsmanagement. Doch was ist wirklich Qualität? Kann Qualität gemessen werden, oder handelt es sich dabei nur um eine subjektive Sichtweise?

Einige meinen hier vermutlich, dass Qualität auf jeden Fall gemessen werden kann. Man nehme die Anzahl der Funktionen und setze sie in Verhältnis dazu, wieviele tatsächlich ordnungsgemäß ihren Dienst verrichten.

Andere mögen dieser Aussage widersprechen. Qualität bedeutet nicht, dass etwas so funktioniert wie es definiert wurde, sondern so, wie es sich der Anwender wünscht (kommt hier etwa das Thema Usability ins Spiel?). Zusätzlich zur angebotenen Funktionalität zählen auch Support, zusätzliche Dienstleistungen und vieles mehr zur Qualität.

Nun kommt die dritte Gruppe und erzählt, dass der Begriff der Qualität aus dem Altertum bekannt ist. Weiters wird im Lateinischen der Begriff *qualitatis* mit der Beschaffenheit (vermutlich eines Gegenstandes?) übersetzt. Weiters wird Qualität nicht nur durch Technik definiert, sondern auch durch eine geistige Haltung. So gesehen kann dies in einer Formel abgebildet werden:

Qualität = Technik + Geisteshaltung (Kamiske)

Durch einen vermehrten Einsatz von **Technik** kann die Qualität verbessert werden. Produkte werden nach demselben Ablauf produziert, manuelle Eingriffe sind oft nicht mehr notwendig und dadurch auftretende Fehler können ausgeschlossen werden. Ein möglicher Diskussionspunkt an dieser Stelle stellt sicherlich (als Beispiel) Bentley dar: Die Lederausstattung wird manuell eingepasst und "montiert". Dies wirkt dementsprechend der obigen Formel entgegen.

Die **Geisteshaltung** stellt jedoch einen sehr wichtigen Baustein zum Gebäude Qualität dar. Steckt kein hohes Maß an Qualitätsbewußtsein in den Köpfen der Mitarbeiter, ist es nahezu unmöglich, qualitativ hochwertige Produkte zu erzeugen. Mitarbeiter müssen diesen Gedanken tragen, sich dafür einsetzen und - wie auch das Management - entsprechende Verbesserungsvorschläge bringen und

durchsetzen (siehe auch Total Quality Management - TQM).

Doch zurück zur Frage: Was ist wirklich Qualität? Qualität wird nicht durch ein Produkt oder ein Unternehmen definiert. Qualität beginnt bei der Zufriedenheit des Kunden und endet beim ersten Rümpfen der Nase. Dies ist klarerweise hart definiert, aber es entspricht in etwa der Erwartungshaltung des Kunden. Und nein! Das ist noch nicht alles: Support, Kommunikation mit dem Kunden, rasche Fehlerbehebung und vieles mehr füllt den Begriff Qualität aus. Es ist also gut zu "sehen", dass es immer und überall Möglichkeiten gibt, die gebotene Qualität zu erhöhen.

Auf den Punkt gebracht: Trotz aller Normen und Richtlinien: Qualität ist eine subjektive Sichtweise. Dieser kann jedoch durch eine effiziente und beharrliche Umsetzung nahegekommen werden.

4.6.2. Enterprise Logging Application Block

Logging muss nicht kompliziert sein. Wer bekannte Logging-Bibliotheken á la NLog nicht verwenden will, oder doch lieber seine Informationen in der Ereignisanzeige sehen möchte, der kann den **Enterprise Logging Application Block** verwenden.

Mittlerweile findet sich im Paket das Tool **Enterprise Library Configuration** mit dem Konfigurationen für die unterschiedlichen **Application Blocks** erstellt werden können. Für mein Beispiel würde dies so aussehen:

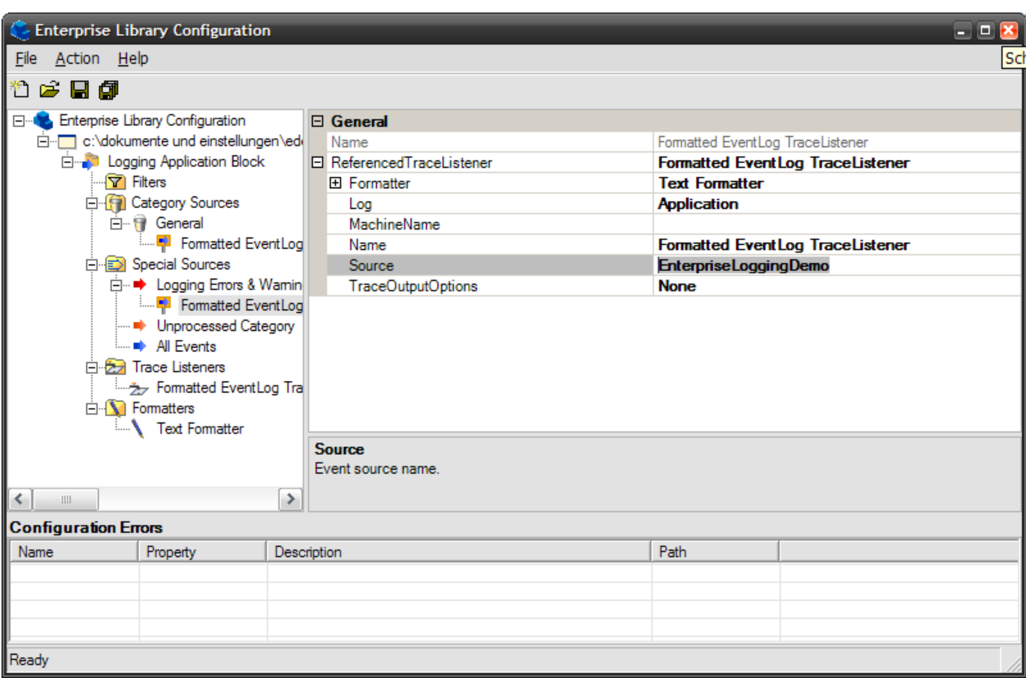


Abbildung 62: Enterprise Library Configuration

Die durch die Konfigurierung entstehende Konfigurations-Datei (welche der .NET Standard Anwendungs-Konfiguration entspricht) muss nun nur mehr in das Anwendungsverzeichnis kopiert werden. Zudem sind folgende Assemblies einzubinden:

```
Microsoft.Practices.EnterpriseLibrary.Logging
```

Soll nun ein Eintrag mitgeschrieben werden, erfolgt dies durch den einfachen Aufruf:

```
Logger.Write("test");
```

Durch die Möglichkeit, unterschiedliche `TraceListener` einzusetzen, müssen die Informationen nicht zwangsweise in die Ereignisanzeige geschrieben werden. Auch Datenbanken und andere Ziele sind möglich. Zusätzlich können viele weitere Einstellungen konfiguriert werden.

Download-Link: Enterprise Library 3.0 January 2007 CTP

PS: Man beachte, dass die **Enterprise Library** zahlreiche weitere **Application Blocks** inkludiert und somit für jeden Entwickler von Bedeutung sind. Ein Blick darauf lohnt sich also.

5. Tools

5.1. Tools: DotLucene - Fulltext Search Engine for .NET

Wer für diverse Anwendungsfälle eine kostengünstige und vor allem schnell Volltextsuche benötigt, der sollte sich auf jeden Fall DotLucene genauer ansehen.

Hinter diesem Open-Source-Projekt verbirgt sich eine vielversprechende Lösung für dieses Thema.

Ein kurzer Auszug aus der Featureliste:

- Gute Performance
- Ranking
- Hervorhebung der Suchbegriffe in den Ergebnissen
- Suche nach Metadaten
- Speicherung von vollindizierten Dokumenten

Durch die Benutzung der Online Demo kann man sich von der Geschwindigkeit und von einigen Features überzeugen.

Im gleichen Atemzug kann man sich auch den - auf der Seite von DotLucene vorgeschlagenen - Indexing Server Seek a File ansehen. Der Server läuft als Windows Dienst und kann folgende und weitere Dateitypen indizieren:

- DOC
- PDF
- XLS
- PPT
- RTF
- HTML
- TXT
- XML

Ein Blick auf diese Tools sollte sich als durchaus lohnen.

5.2. GUI für Windows Installer XML (WiX)

Windows Installer XML dürfte dem einen oder anderen durchaus bekannt sein.

The Windows Installer XML (WiX) is a toolset that builds Windows installation packages from XML source code. The toolset supports a command line environment that developers may integrate into their build processes to build MSI and MSM setup packages.

Nun, auch hier wird es ähnlich wie bei Sandcastle mit der Konfiguration ein wenig zeitintensiver. Dafür gibt es aber Abhilfe.

CalmWix. Es gibt zwar einige grafische Oberflächen für WiX, aber CalmWix verspricht, die funktionsreichste zu sein bzw. generell zu werden. Ein erster Screenshot zeigt die ansprechende Oberfläche (im Stil von Visual Studio 2005 gehalten) und lässt auf wesentlich mehr hoffen.

Ein Release gibt es noch nicht, aber wer möchte, kann sich den Sourcecode genauer ansehen.

5.3.Sandcastle Helferleins

Wer Sandcastle als Ersatz für beispielsweise NDoc einsetzt oder einsetzen möchte, dem sei ein Blick auf zwei Helferlein zu empfehlen:

Sandcastle Help File Builder
Eine grafische Oberfläche, welche die Bedienung und Konfiguration von Sandcastle vereinfacht.

Sandcastle Add-in
Add-in für Visual Studio 2005.

Und wer noch immer nicht genug hat, der kann noch einen Blick auf die MSBuild Scripts für Sandcastle werfen.

5.4.ReSharper UnitRun: kostenloser Testdriven.NET Gegenspieler

Von JetBrains gibt es ein kostenloses Visual Studio Add-In mit dem Unit Tests ausgeführt werden können: ReSharper TestRun [1].

Unterstützt werden folgende Testing-Frameworks:

- NUnit
- csUnit

Ein näherer Blick auf dieses Tool lohnt sich auf jeden Fall.

[1] UnitRun Homepage

5.5. CCTray: CruiseControl.NET Build-Fortschritt im Überblick

CCTray [1]. Damit lässt sich der Build Progress überwachen und das Tool erlaubt es, in einige Operationen einzugreifen.

[1] CCTray Website

5.6. CCNetConfig: CruiseControl.NET Konfigurationsdateien einfach erstellt

Wer eine CruiseControl.NET [1] Installation sein Eigen nennt, dem dürfte CCNetConfig [2] als Unterstützung hilfreich sein. Mittels grafischer Oberfläche können damit CC.NET Konfigurations-Dateien einfachst erstellt werden.

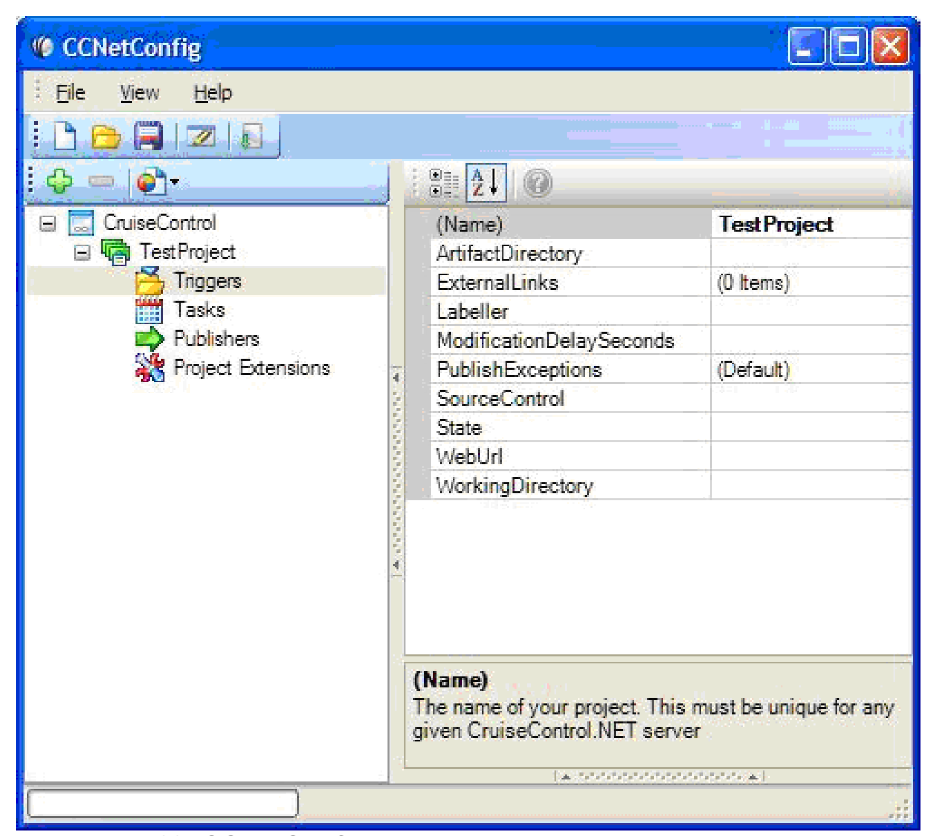


Abbildung 63: CCNetConfig

[1] CruiseControl.NET
[2] CCNetConfig

5.7.WMI Code Creator

Mehr zufällig als gewollt, fand ich gestern ein kleines Tool von Microsoft zur Generierung von Code für die Arbeit mit der WMI. Dieses Tool erlaubt es in einer GUI WMI Namespaces und Klassen auszuwählen und erstellt entsprechend der Auswahl passenden Sourcecode.

Folgende Sprachen werden unterstützt:

- C#
- Visual Basic .NET
- Visual Basic Script (VBS)

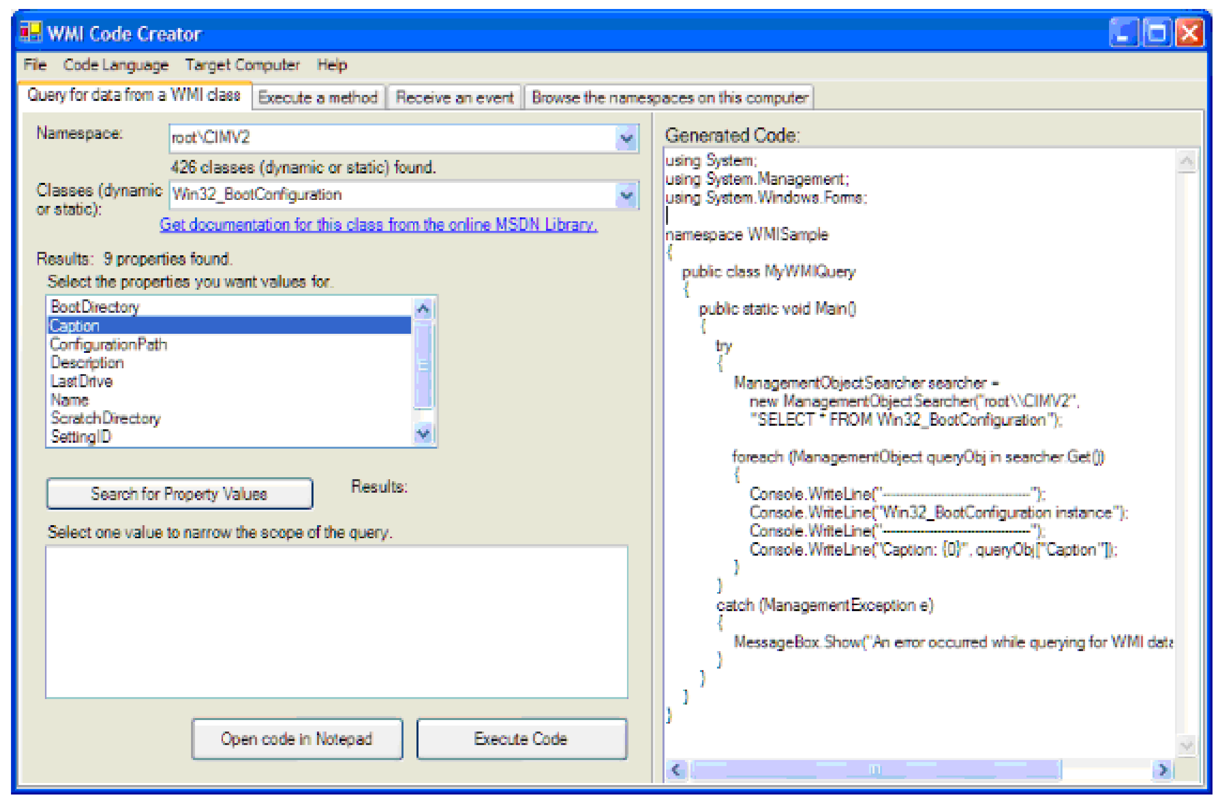


Abbildung 64: WMI Code Creator

Weitere Features

- Browsing namespaces
- Method execution
- Support of remote computer
- and more …

Download WMI Code Creator 1.0

5.8.Guidance Explorer

Für die Entwicklung unter .NET gibt es ja viele hilfreiche Tools und Anwendungen. Den Guidance Explorer muss man hier jedoch immer wieder einmal erwähnen.

Was bietet dieses Tool? Nun ja, Guidelines, Patterns, Anti-Patterns, Checklists, Code Samples, und Test Cases. Das Angebot ist wirklich gut, wird ständig erweitert und sollte bei keinem .NET Programmierer fehlen.

5.9.Documentation Generator: CodeDoc

Aus Zufall bin ich heute auf den Documentation Generator CodeDoc [1] gestoßen. Das Teil ist Freeware und scheint die notwendigsten Funktionalitäten mitzubringen. Die Demo sieht mal ganz gut aus. Ein Test wird sich wohl nicht vermeiden lassen. Wer also noch auf der Suche nach einem entsprechenden Tool ist, der könnte sich CodeDoc durchaus mal genauer ansehen.

[1] CodeDoc Documentation Generator

5.10.LINQ - Kennst du schon?

Vergangenes Wochenende habe ich mich ein wenig mit LINQ beschäftigt. Mit Hilfe dieser Erweiterung lassen sich sehr einfach Datenquellen abfragen. LINQ erweitert C# und VB.NET um eine native Sprach-Syntax, Datenquellen abzufragen. Von LINQ gibt es zwei unterschiedliche Varianten:

Mit **DLINQ** können relationale Datenquellen abgefragt werden. Die zweite Variante nennt sich **XLINQ** und ermöglicht die Abfrage von XML-Daten.

Und so sieht zum Beispiel eine DLINQ-Abfrage aus:

```
public void Linq7()
{
    List products = GetProductList();

    var productNames =
        from p in products
        select p.ProductName;



    Console.WriteLine("Product Names:");
    foreach (var productName in productNames)
    {
        Console.WriteLine(productName);
    }
}
```

Weitere Informationen zu LINQ sind auf der LINQ Project Homepage zu finden.

5.11.ASP.NET Deployment Tool

Visual Studio 2005 bietet von Haus aus schon sehr gute Möglichkeiten eine ASP.NET Anwendung zu veröffentlichen. Allerdings gibt es keine Möglichkeit ein bestimmtes Deployment Schema zu speichern oder festzulegen.

Das ASP.NET Deployment Tool von Andreas Kraus bietet genau dieses Feature. Hat man erst mal ein Projekt angelegt, gibt es verschiedene mögliche Optionen.

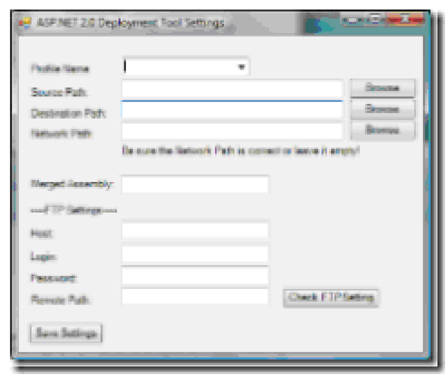


Abbildung 65: ASP.NET Deployment Tool 1

Neben dem einfachen lokalen Deployment ist es möglich die Dateien per Netzwerkpfad oder FTP zu veröffentlichen. Außerdem kann eingestellt werden, ob das Zielverzeichnis gelöscht, die `Web.Config` übertragen oder alle Assemblies in ein einziges zusammengefasst werden sollen.

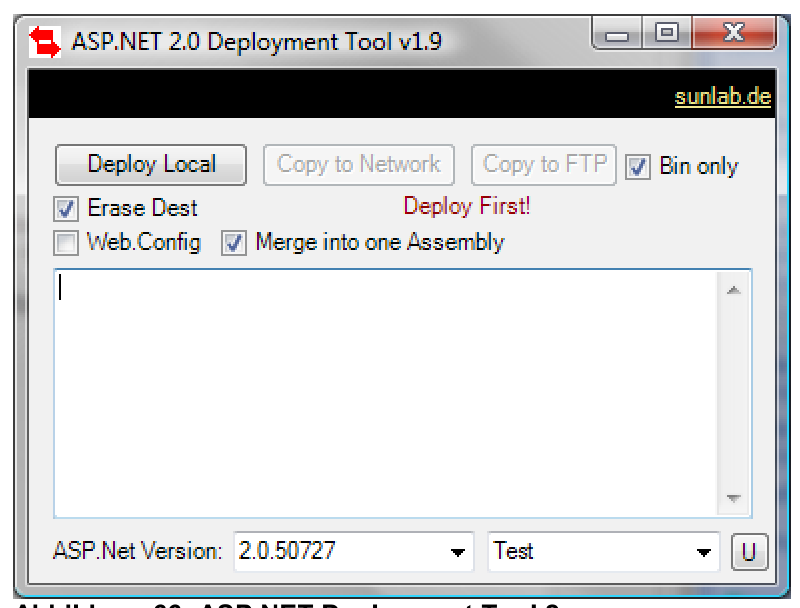


Abbildung 66: ASP.NET Deployment Tool 2

Auch wenn das Programm selbst noch mit kleinen Bugs zu kämpfen hat, ist es eine sehr gute Alternative zum Visual Studio integrierten Deployment.

Mehr Informationen und den Download gibt es unter http://www.sunlab.de/Tools.aspx

5.12.Lokalisierung des Community Servers

Nach erfolgreicherer Installation des Community Servers, folgt in den meisten Fällen die Lokalisierung. Zum Glück gibt es bereits ein deutsches Language-Pack, deren Installation hier kurz erläutert werden soll.

Den Download des Language-Packs für die Version 2.1 findet man auf den offiziellen Seiten im Downloadbereich. Der direkt Link lautet: http://communityserver.org/files/folders/language_

Allerdings ist hierfür eine Registrierung notwenig.

In dem herunter geladenen ZIP-Archiv findet man neben der `ReadMe.txt` den Order `de-DE`, in dem die Sprachdateien enthalten sind. Dieser Order muss per FTP in den Ordner `Languages` des Community Server Webs übertragen werden. Anschließend sollte man von folgenden Dateien ein Backup anlegen:

```
\Languages\languages.xml
\communityserver.config
web.config
```

Zunächst muss Datei `\Languages\languages.xml` abgeändert werden. Folgende Zeile muss in die `<root>` Section eingefügt werden:

```
<language name="German" key="de-de" />
```

Diese Zeile gibt an, dass nun eine weitere Sprache in dem Ordner `de-de` vorhanden ist.

Möchte man die Standardsprache des Community Server auf Deutsch festlegen, muss eine Zeile in der Datei `\communityserver.config` geändert werden. Das Attribut `defaultLanguage` des *Core* Nodes enthält per default den Wert `en-US`. Dieser muss mit dem Wert `de-DE` ersetzt werden.

```
<Core defaultLanguage="de-DE"
```

Nun muss nur noch die Ausgabe der Uhrzeiten und des Datums angepasst werden, da diese weiterhin im US Format ausgegeben werden. Hierfür ist es nötig die `Web.config` zu ändern. Falls folgende Zeile nicht bereits vorhanden ist, muss sie eingefügt, ansonsten abgeändert werden. Wichtig ist die Platzierung in der `<system.web>` Section.

```
<globalization culture = "de-DE"></globalization>
```

Nachdem die `Web.Config` gespeichert wurde, ist die Lokalisierung abgeschlossen und der Community Server sollte nun die deutsche Sprache verwenden.

5.13.Community Server Installation HowTo



Abbildung 67: Community Server Installation 1

Sucht man eine ASP.NET Foren-Software trifft man schnell auf dem Community Server, der von der Firma telligent entwickelt wurde. Die Feature Liste des Community Server ist lang. Nach der Installation erhält man nicht nur ein einfaches Forum, sondern vielmehr eine Kommunikations-Plattform mit zusätzlicher Unterstützung von Weblogs und Image Galleries, also alles was eine moderne Community Site ausmacht.

Neben der kostenlosen Express Edition, ist es möglich eine Standard, Professional oder Enterprise Version mit mehr Features zu erwerben. Eine genaue Übersicht der Versionen findet man unter der URL https://store.telligent.com/FeatureMatrix.aspx#Edi...

In diesem Artikel wird die Version 2.1.61025.2 des Community Servers für ASP.NET 2.0 per Web installiert. Grundsätzlich ist hierfür zunächst nur der Zugriff auf das Web per FTP nötig. Der Download des Community Server ist nach erfolgreicher Anmeldung unter der URL http://www.communityserver.org möglich. Neben der Installation per Web-Oberfläche ist es ebenfalls möglich (entsprechende Rechte vorausgesetzt) mit dem Windows Installer die zukünftige Community einzurichten.

Vorbereitung

Nachdem der Download erfolgreich war und das ZIP-Archiv entpackt wurde, findet man zwei Ordner mit den Namen `Web` und `SqlScripts` vor. In diesem Beispiel wird eine neue Community eingerichtet, somit kann der Inhalt des Ordners `SqlScripts` vernachlässigt werden. Aktualisiert man eine vorhandene Installation, findet man dort die entsprechenden Scripte für die Aktualisierung der Datenbank.

Zunächst muss in der Datei `/web/installer/default.aspx` die Variable `INSTALLER_ENABLED` auf `true` gesetzt werden. Diese Einstellung aktiviert die Installation per Web-Oberfläche.

```
bool INSTALLER_ENABLED = true;
```

Anschließend kann der Inhalt des Web-Ordners per FTP auf den gewünschten Server übertragen werden.

Zusätzlich zur Übertragung der Dateien per FTP, muss eine SQL-Server Datenbank eingerichtet werden. Während der Web-Installation werden der Datenbankname, der Benutzername und das Passwort benötigt. Zusätzlich muss der Benutzer den Datenbank-Rollen `public` und `db_owner` zugeordnet werden.

Installation

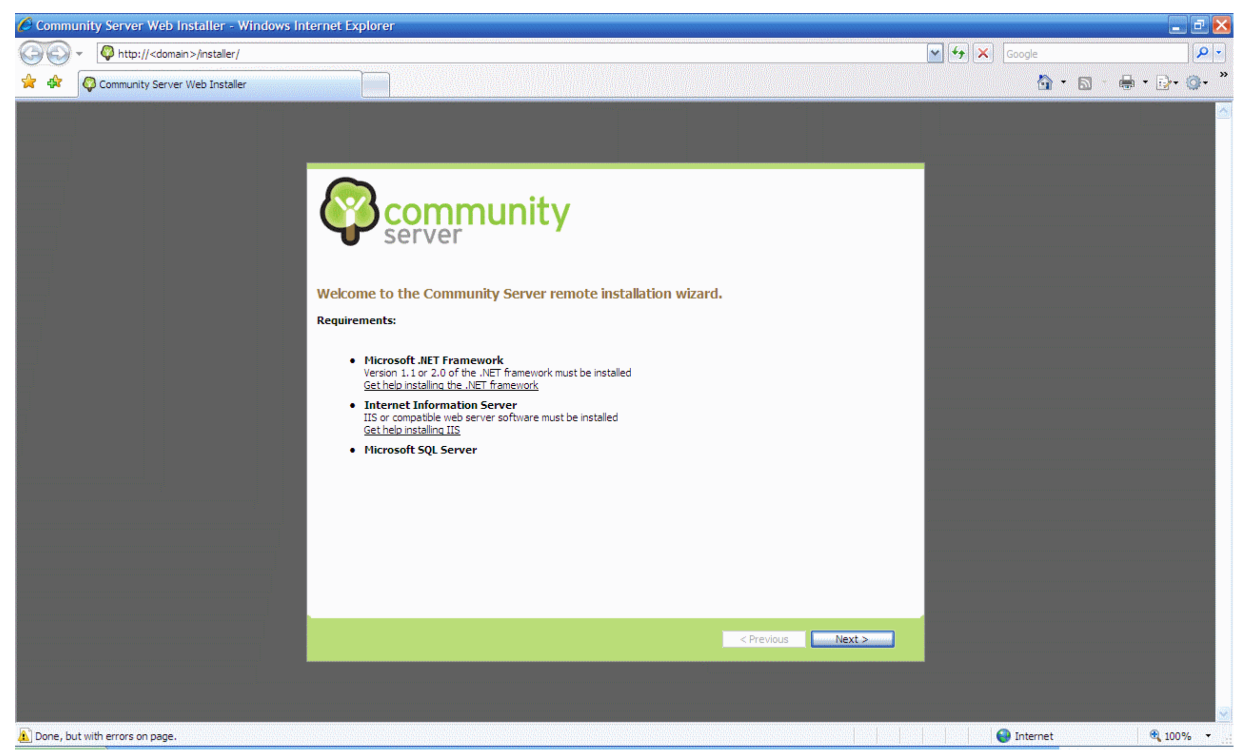


Abbildung 68: Community Server Installation 2

Der Aufruf der URL http://<domain>/installer startet den Web-Installer. Nachdem den Lizenzbestimmungen zugestimmt wurde, muss die Verbindung zur Datenbank angelegt werden. Hierfür wird wie oben bereits geschrieben, der Benutzername der Datenbank und das Passwort benötigt, außerdem muss festgelegt werden, ob sich die Anwendung per Windows oder Sql Server Authentication verbinden soll. Da in diesem Artikel ein Datenbank-Benutzer angelegt wurde, fällt die Wahl auf "SQL-Server Authentication".

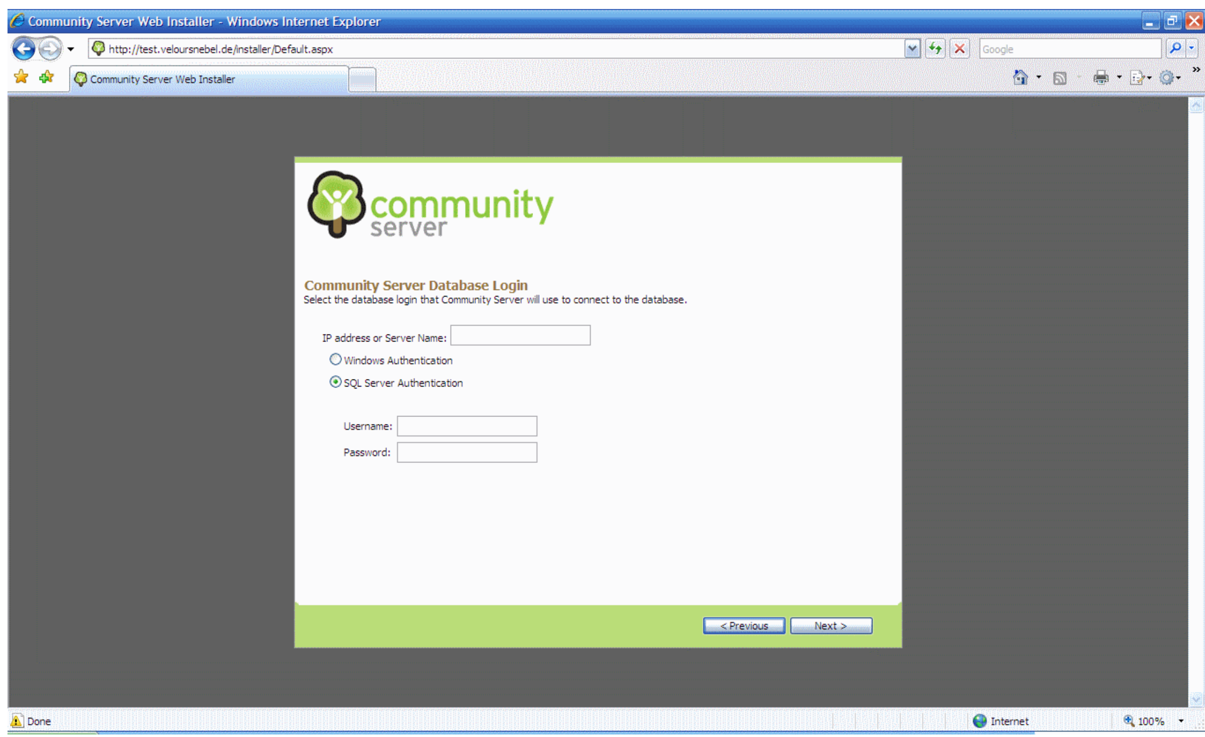


Abbildung 69: Community Server Installation 3

Der Installer versucht nun eine Verbindung mit der Datenbank herzustellen. War dies erfolgreich, kann die gewünschte Datenbank ausgewählt werden.

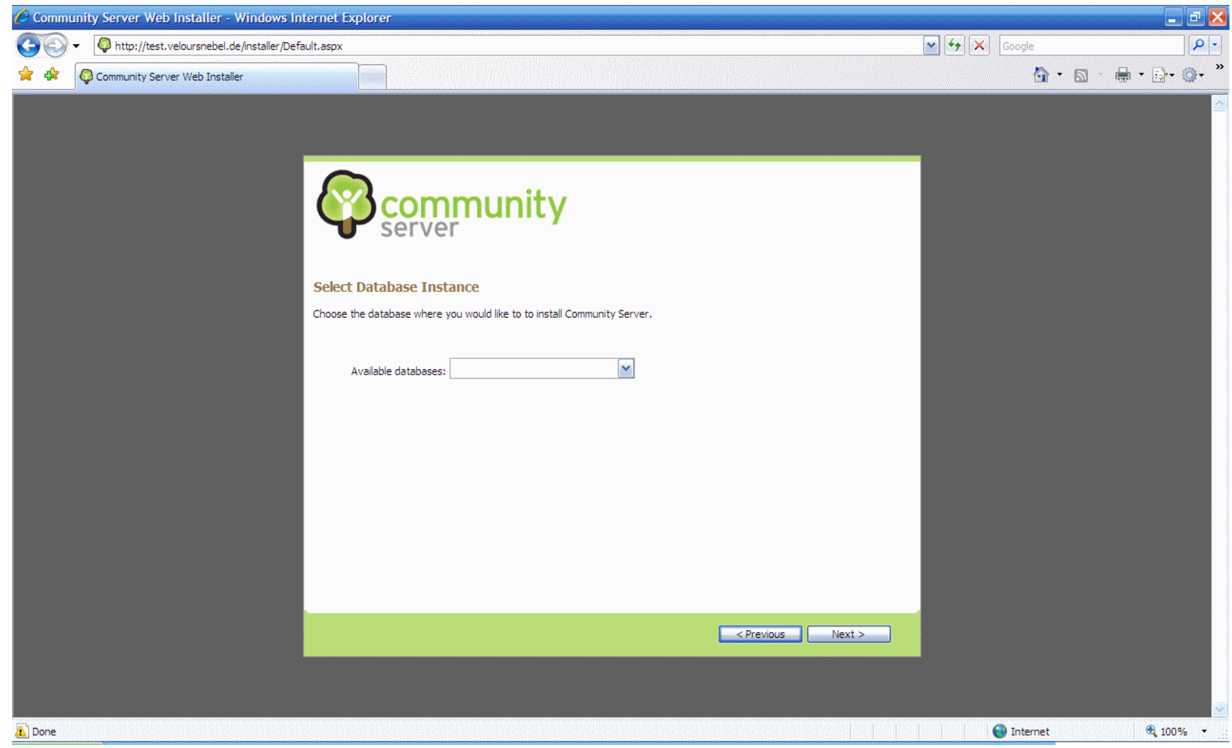


Abbildung 70: Community Server Installation 4

Nun können verschiedene Optionen ausgewählt werden, die die Einrichtung der Datenbank beeinflussen.

Script ASP.NET MemberRoles
Diese Option installiert die ASP.NET Memberroles. Da lediglich der Community Server die Datenbank verwendet, kann diese Option aktiviert bleiben. Nutzen mehrere Anwendungen die ASP.NET Memberroles, muss diese Option deaktiviert werden.

Script Community Server
Ist diese Option aktiviert, wird das Standard Community Server Schema installiert. Bei einer Neuinstallation wird diese Option natürlich benötigt.

Create Community
Auch diese Option wird benötigt, wenn eine Neuinstallation statt findet und installiert eine neue Community.

Im nächsten Schritt wird der erste Benutzer der Community angelegt, der gleichzeitig auch die Aufgabe des Administrators übernimmt. Neben der URL der Community, wird ein Username, ein Passwort, und die E-Mail Adresse für den zukünftigen Administrator benötigt. Außerdem wird in diesem Beispiel die Option *Create Sample Data* ausgewählt. Benötigt man die Beispieldaten nicht, kann man diese Option natürlich auch deaktivieren.

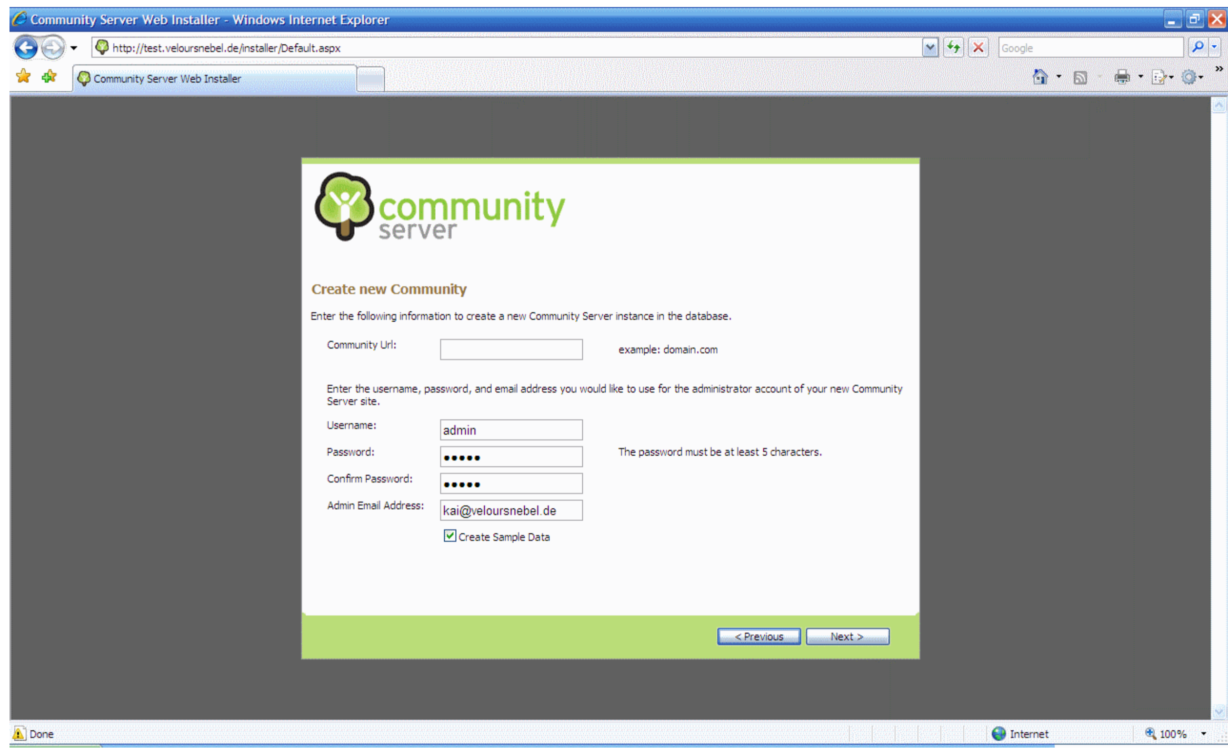


Abbildung 71: Community Server Installation 5

Nachdem der Installer fertig gearbeitet hat, ist die Installation abgeschlossen. Ist bereits eine `Web.config` in dem Web vorhanden zeigt der Installer die nötigen Informationen an, die dann per Copy & Paste nachgepflegt werden können. Aus Sicherheitsgründen sollte das Verzeichnis `Installer` anschließend gelöscht werden.

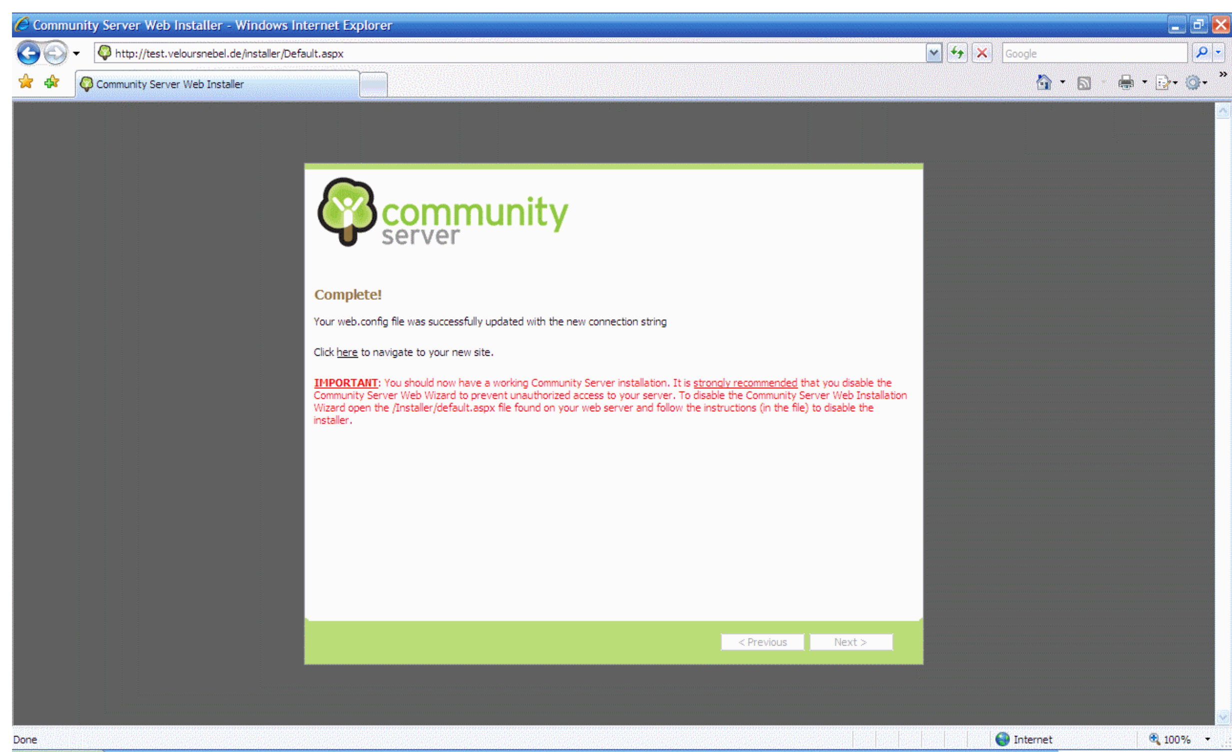


Abbildung 72: Community Server Installation 6

Installation abgeschlossen

Sofern bei der Installation die Option *Create Sample Data,* findet man nun einen bereits eingerichteten Weblog, eine eingerichtete Image Gallery und natürlich das Forum mit Test-Einträgen vor. Nachdem man sich als Administrator eingeloggt hat ist es möglich über den Menüpunkt Control Panel, die weitere Konfiguration der Community vorzunehmen.

Weitere Informationen zur Installation und Konfiguration des Community Server findet man unter der URL http://docs.communityserver.org/. Außerdem ist ein weiteres deutsches Tutorial von Alex Bierhaus unter der URL

http://www.codezone.de/DetailPage.Codezone?GUID=b6454095-59bf-474d-be09-cb68a126c424

verfügbar.

Erfolgreich eingesetzte Installation findet man unter folgenden Adressen, insbesondere erwähnenswert ist das offizielle ASP.NET und XBox Forum von Microsoft.

- www.hive.net
- Weblogs.msdn.com
- forums.asp.net
- forums.xbox.com
- www.glengamoi.com

5.14.SQL Server Web Data Administrator

Es gibt nicht nur den "ASP.Net EnterpriseManager[1]" um per Web-Frontend SQL Server Datenbank zu administrieren, sondern auch den "SQL Server Web Data Administrator[2]" von Microsoft.

[1] http://www.aspenterprisemanager.com/
[2] http://www.microsoft.com/downloads/details.aspx?FamilyID=C039A798-C57A-419E-ACBC-2A332CB7F959&displaylang=en

5.15.ILMerge im Einsatz

Mit Hilfe des ILMerge-Tools können mehrere Assemblies zu einer einzigen Assembly zusammengefügt werden. Vor allem bei größeren Projekten entsteht nicht nur eine einzige ausführbare Datei. Durch die Kapselung der Funktionalität in unterschiedliche Bibliotheken werden diese als eigenständige Assemblies abgelegt.

In einigen Fällen kann es nun erwünscht sein, diese Dateien in eine einzige zusammen zu fassen. Ein Paradebeispiel wäre die Vereinfachung des Deployment-Prozesses, da in diesem Fall lediglich eine "Anwendungsdatei" berücksichtigt werden muss (abgesehen von anderen Dateien, Einstellungen, etc. die ausgeliefert werden müssen).

Als Beispiel dient eine simple Anwendung, die nur eine Funktionalität besitzt: das Multiplizieren zweier Integer-Werte. Dafür wurde zusätzlich zur Windows Forms Anwendung eine Klassenbibliothek angelegt. Darin enthalten ist die Klasse `Calculator`.

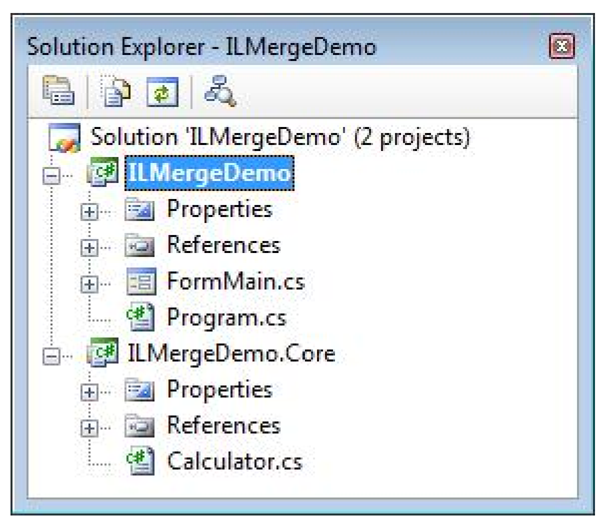


Abbildung 73: ILMergeDemo - Solution Explorer

Nach einem Build im Release-Modus werden die Dateien **ILMergeDemo.exe** und **IlMergeDemo.Core.dll** angelegt. Diese fassen wir nun zu einer einzigen Assembly zusammen. Dies geschieht mit dem folgenden Aufruf:

```
ILMerge.exe /t:winexe /out:ILMergeDemoMerged.exe ILMergeDemo.exe
ILMergeDemo.Core.dll
```

ILMerge selbst ist eine Konsolen-Anwendung, die mittels Parameter gesteuert werden kann:

/t:filename bzw. **/target:filename**
Hiermit wird eingestellt, ob es sich um eine Windows Anwendung, eine Konsolenanwendung oder eine Klassenbibliothek handelt. Wird dieser Parameter nicht angegeben, entspricht der Ausgabetyp dem Typ der ersten Assembly (Primary Assembly), die der Ausgabe hinzugefügt wird.

/out:filename
Dadurch wird der Name des Ziels angegeben.

/v1 oder **/v1.1** oder **/v2**
Diese Option ist nur bei ILMerge für das .NET Framework 2.0 verfügbar. Damit kann eine Assembly gelinkt werden, die auch unter einer anderen Framework-Version lauffähig ist.

/log:filename
Für eine automatisierte Verwendung bietet sich der Parameter /log an. Durch ihn kann die Ausgabe des Vorgangs in eine anzugebende Logdatei geschrieben werden. Etwaige Fehler können dadurch zu einem späteren Zeitpunkt gefunden und analysiert werden.

Natürlich können noch weitere Parameter angegeben werden. Diese können der Dokumentation zu ILMerge entnommen werden (ist im Download enthalten).

Vergleich mittels Reflector for .NET

Sehen wir uns die Beispielanwendung in Lutz Roeder's Reflector for .NET an, sind die beiden Assemblies extra ausgewiesen:

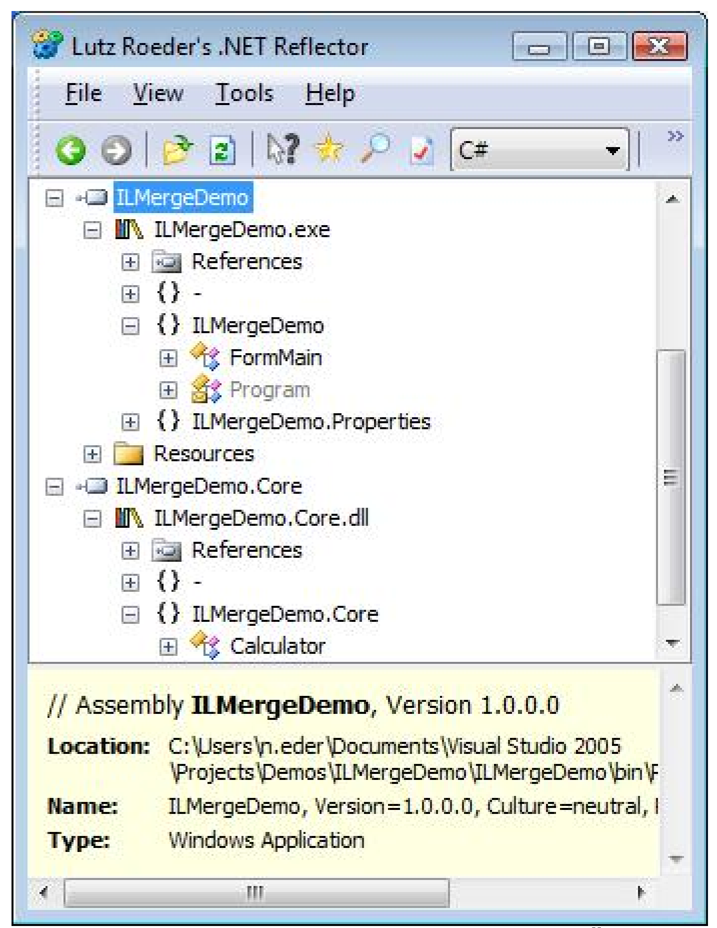


Abbildung 74: Lutz Roeder's .NET Reflector - Übersicht

Nach dem Merge-Vorgang ist ersichtlich, dass sich alles in einer einzigen Assembly befindet:

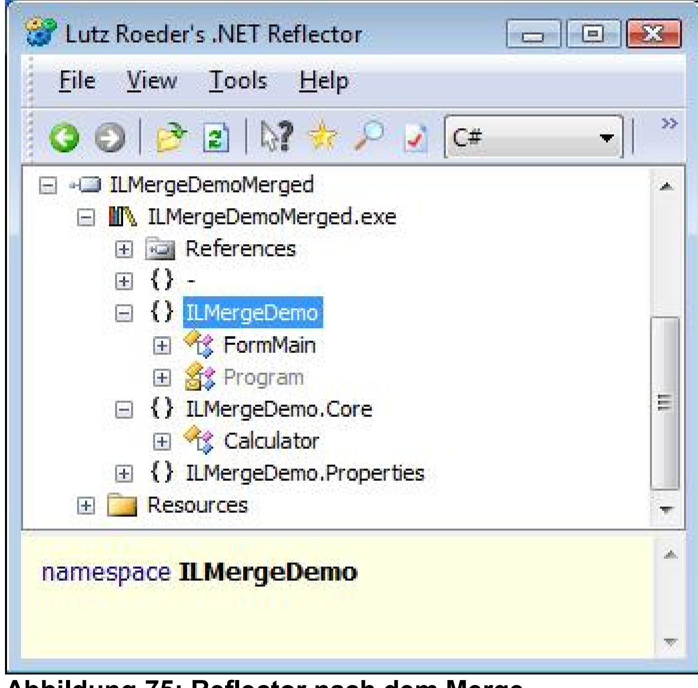


Abbildung 75: Reflector nach dem Merge

Was passiert mit den Ressourcen?

ILMerge kann Ressourcen nicht zusammenführen, sondern lediglich kopieren, was bei der Ausführung auch passiert. Nun kann es bei der Laufzeit der zusammengeführten Assembly zu Fehlern kommen. Dies ist beispielsweise dann der Fall, wenn die Ressourcen Referenzen zu Typen kodieren. Diese können danach nicht mehr aufgelöst werden: Die Referenz verweist auf die ursprüngliche Assembly, die es nach dem Merge-Vorgang nicht in der "Sammel-Assembly" gibt.

Kann ich ILMerge direkt in ein Projekt einbinden?

Mit Visual Studio 2005 ist es möglich, ausführbare Assemblies als Referenz einzubinden. Dadurch kann ILMerge.exe in ein anderes Projekt quasi als Klassenbibliothek eingebunden werden.

Weitere Informationen

Kommerzielle Nutzung: Eine kommerzielle Nutzung von ILMerge ist laut Lizenz möglich.

PDB-Dateien zusammenführen: ILMerge kann auch PDB-Dateien (Debug-Informationen und Projektstatus) zusammenführen. Hierbei ist zu beachten, dass ILMerge für .NET 2.0 nur 2.0er PDB-Dateien zusammenführen kann. Jedoch können Assemblies früherer Versionen zusammengeführt werden. Mit ILMerge für das .NET Framework 1.1 können auch PDB-Dateien der 1.1er Version zusammengeführt werden.

Mono und Rotor: Bis zum aktuellen Zeitpunkt unterstützt ILMerge weder Rotor noch Mono.

GUI: Für ILMerge gibt es zur Erleichterung grafische Oberflächen von diversen Anbietern. Eine kostenlose Variante ist NuGenUnify.

Die aktuelle Version kann über das Microsoft Download Center bezogen werden.

5.16. Microsoft SQL Server Database Publishing Wizard 1.0

Durch den SQL Server-Datenbankveröffentlichungs-Assistent können Datenbanken in T-SQL-Skripts oder direkt an Hostingdienstanbieter veröffentlicht werden.

Der SQL Server Database Publishing Wizard ermöglicht die Bereitstellung von SQL Server-Datenbanken in einer gehosteten Umgebung an einen Server mit SQL Server 2000 oder 2005. Dabei wird eine einzelne SQL-Skriptdatei generiert, die zum Neuerstellen einer Datenbank (sowohl Schema als auch Daten) in einer

freigegebenen, gehosteten Umgebung verwendet werden kann, bei der die einzige Konnektivität mit einem Server durch einen webbasierten Steuerungsbereich mit einem Skriptausführungsfenster besteht. Der SQL Server Database Publishing Wizard kann Datenbanken auch direkt auf Server hochladen, die sich beim freigegebenen Hostinganbieter befinden, sofern diese Funktionalität vom Hostingdienstanbieter unterstützt wird.

Der SQL Server Database Publishing Wizard kann optional auch direkt in Visual Studio 2005 und/oder Visual Web Developer 2005 integriert werden. Nach erfolgter Integration können Datenbanken aus der Entwicklungsumgebung heraus auf einfache Weise veröffentlicht werden.

Und hier die Screenshots des Tools, als auch die ersten Ergebnisse:

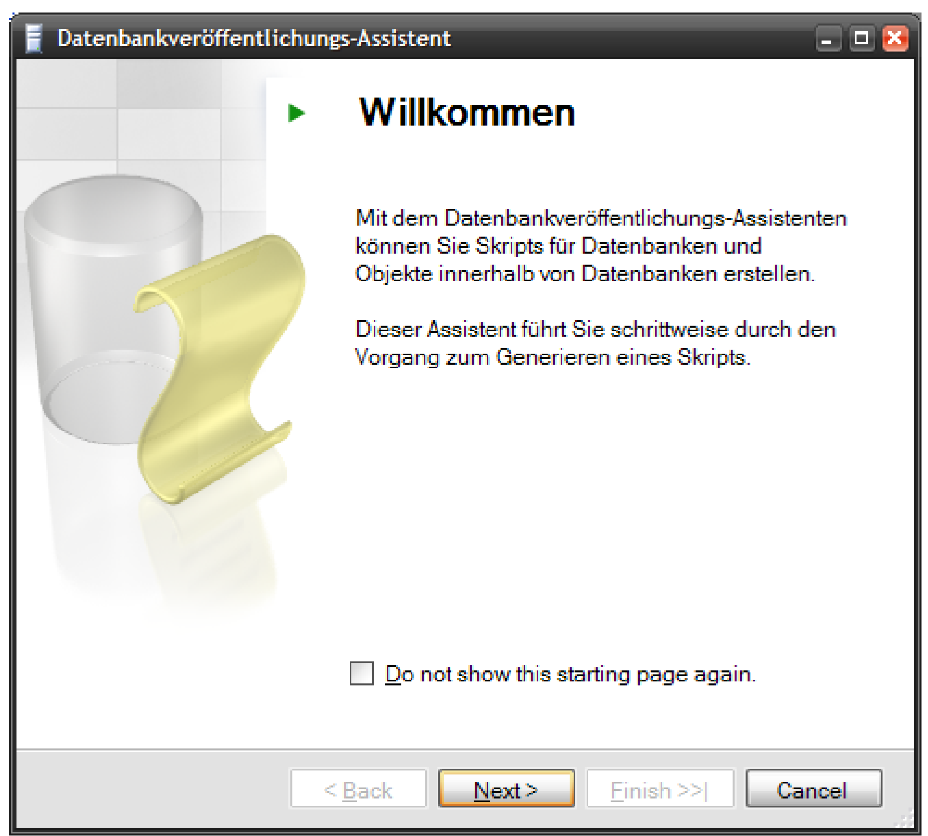


Abbildung 76: Database Publishing Wizard 1

Datenbankveröffentlichungs-Assistent

Quellserver auswählen

Wählen Sie den Server aus, der die zu veröffentlichende Datenbank enthält.

Serverinformationen angeben

Server (local)\SQLExpress

Authentifizierung

Windows-Authentifizierung verwenden

SQL Server-Authentifizierung verwenden

Benutzername

Kennwort

Verbindungszeichenfolge bereitstellen

Verbindungszeichenfolge

< Back Next > Finish >>| Cancel

Abbildung 77: Database Publishing Wizard 2

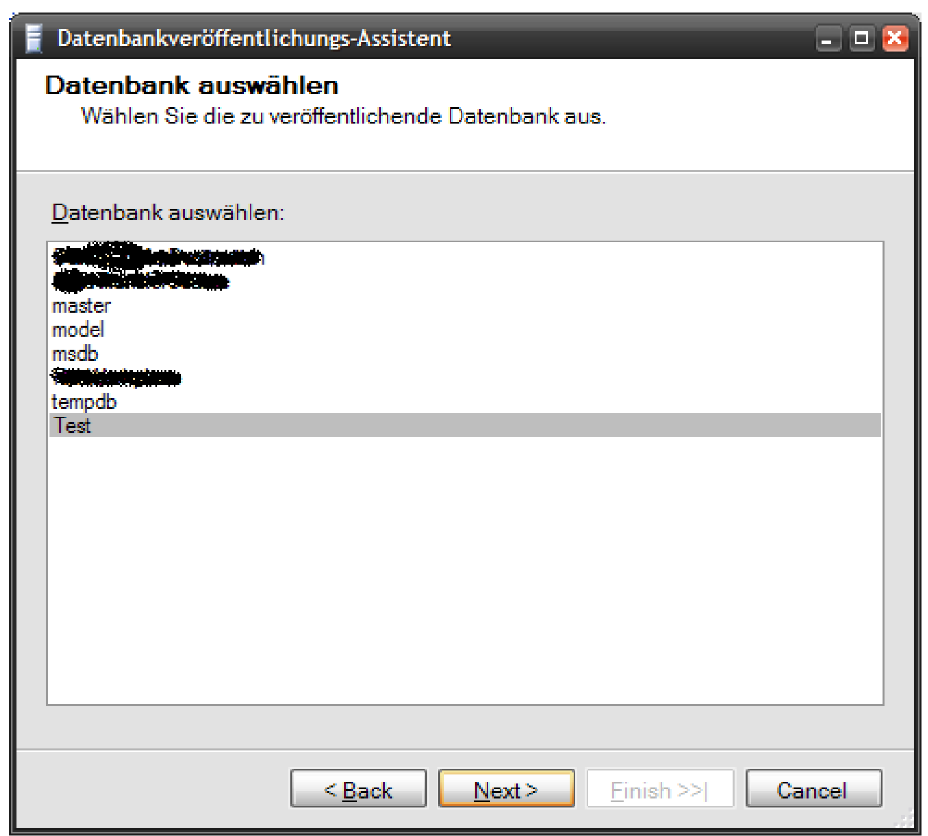


Abbildung 78: Database Publishing Wizard 3

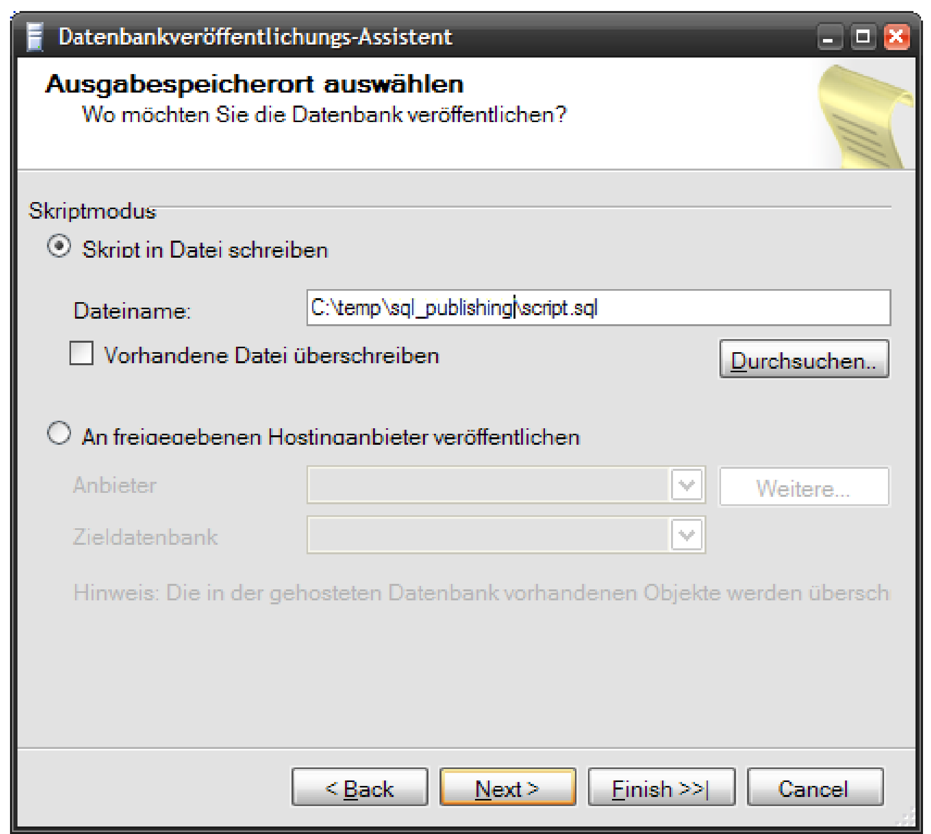


Abbildung 79: Database Publishing Wizard 4

An dieser Stelle könnte der angesprochene Hostinganbieter eingetragen werden.

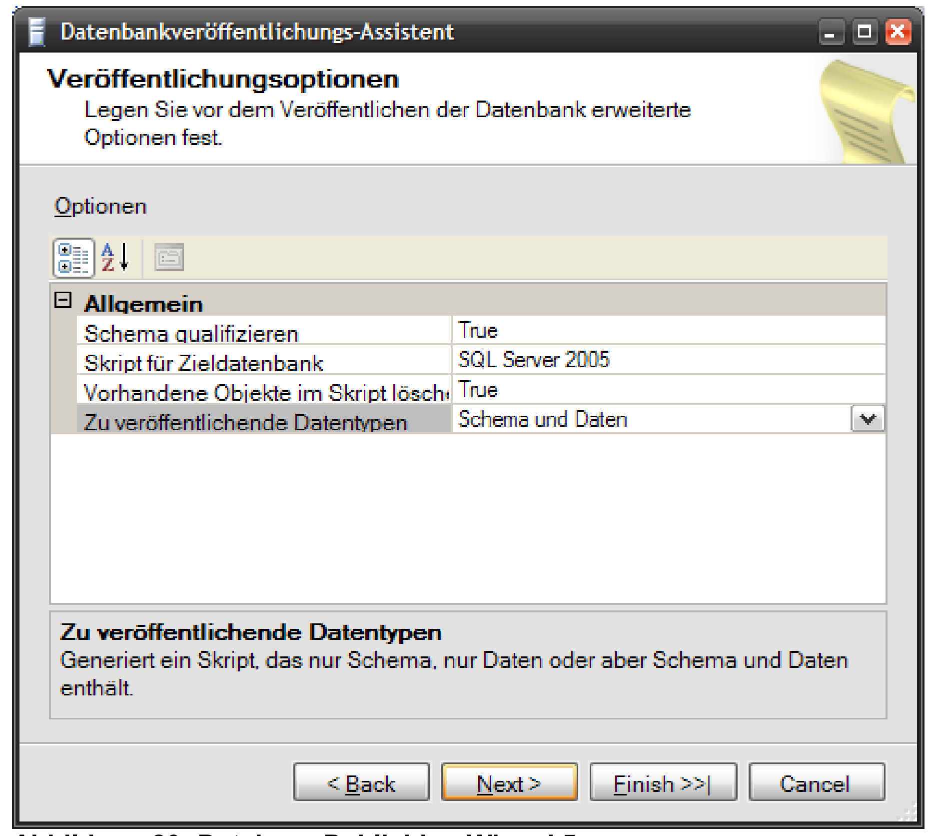


Abbildung 80: Database Publishing Wizard 5

Als Zieldatenbank können **SQL Server 2000** und **SQL Server 2005** ausgewählt werden. Bei den Datentypen können **Schema und Daten**, **Schema** bzw. **Daten** eingestellt werden.

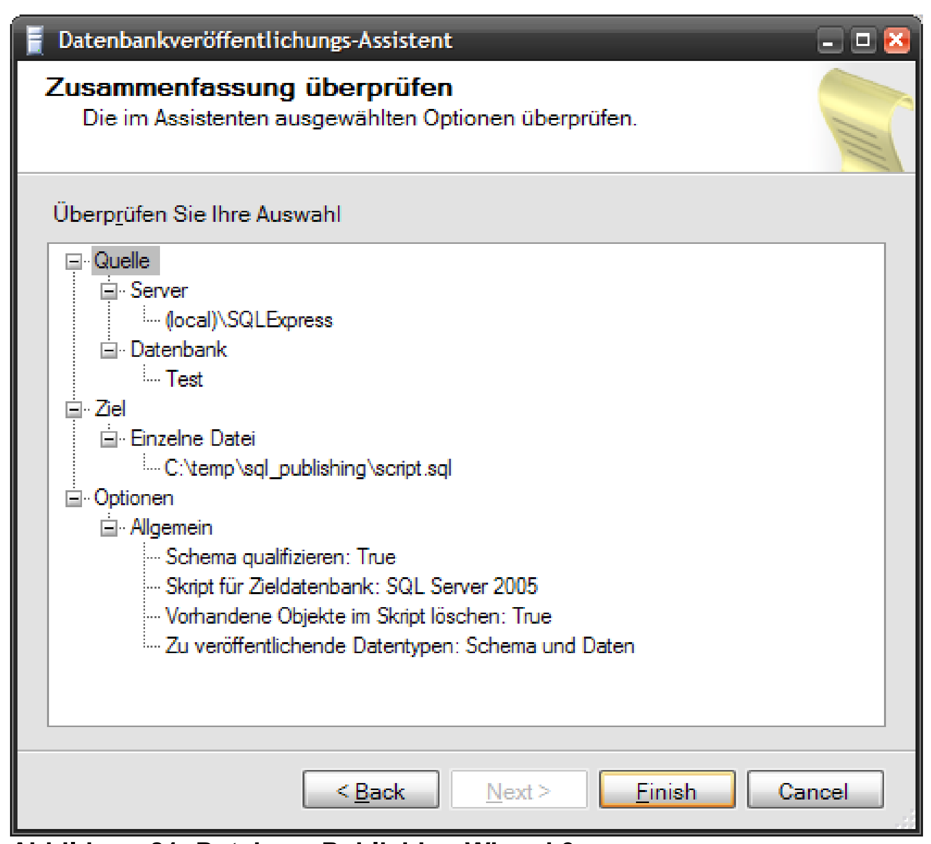


Abbildung 81: Database Publishing Wizard 6

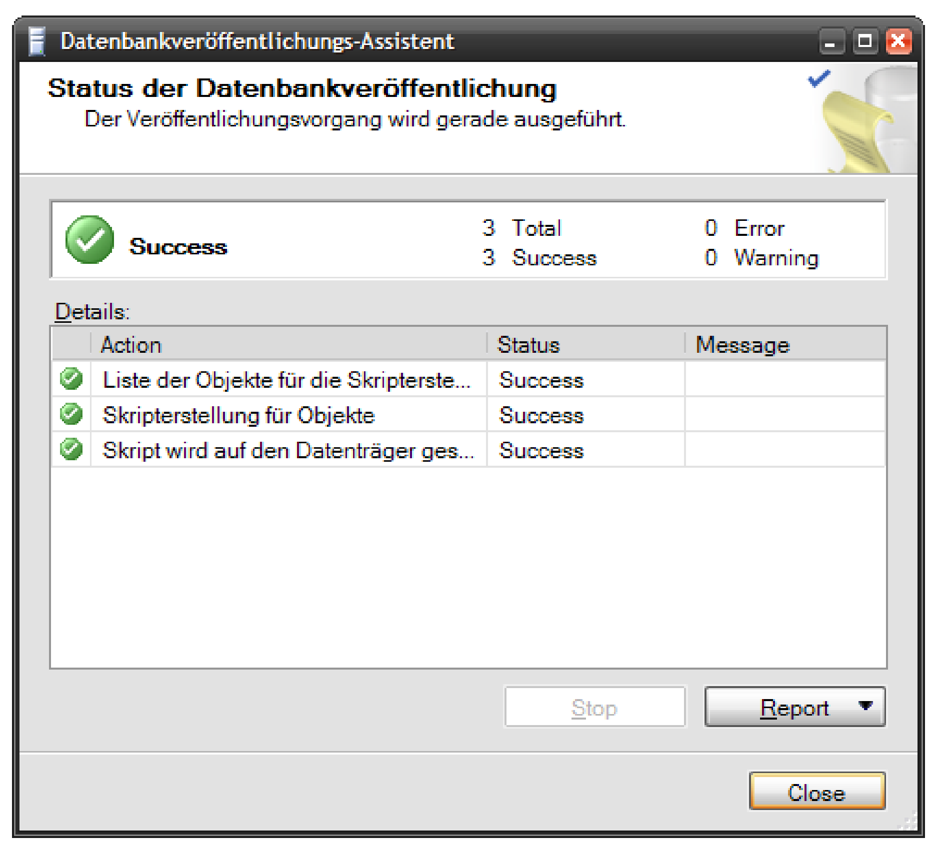


Abbildung 82: Database Publishing Wizard 7

Die Ausgabe dieser Testdatenbank sieht dann folgendermaßen aus:

```
/****** Object:  Table [dbo].[tPerson]
 * Script Date: 01/30/2007 11:24:15 ******/
IF  EXISTS (SELECT * FROM sys.objects
WHERE object_id = OBJECT_ID(N'[dbo].[tPerson]')
AND type in (N'U'))
DROP TABLE [dbo].[tPerson]
GO
/****** Object:  Table [dbo].[tProperty]
 * Script Date: 01/30/2007 11:24:15 ******/
IF  EXISTS (SELECT * FROM sys.objects
WHERE object_id = OBJECT_ID(N'[dbo].[tProperty]')
AND type in (N'U'))
DROP TABLE [dbo].[tProperty]
GO
/****** Object:  Table [dbo].[tPersonAge]
 * Script Date: 01/30/2007 11:24:15 ******/
IF  EXISTS (SELECT * FROM sys.objects
WHERE object_id = OBJECT_ID(N'[dbo].[tPersonAge]')
AND type in (N'U'))
DROP TABLE [dbo].[tPersonAge]
GO
/****** Object:  Table [dbo].[tPersonAge]
 * Script Date: 01/30/2007 11:24:15 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
IF NOT EXISTS (SELECT * FROM sys.objects
WHERE  object_id = OBJECT_ID(N'[dbo].[tPersonAge]')
AND type in (N'U'))
BEGIN
```

```
CREATE TABLE [dbo].[tPersonAge](
    [PersonID] [int] NULL,
    [Age] [int] NULL,
    [Test] [image] NULL
)
END
GO
/****** Object:  Table [dbo].[tProperty]
 * Script Date: 01/30/2007 11:24:15 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
IF NOT EXISTS (SELECT * FROM sys.objects
WHERE object_id = OBJECT_ID(N'[dbo].[tProperty]')
AND type in (N'U'))
BEGIN
CREATE TABLE [dbo].[tProperty](
    [ID] [int] IDENTITY(1,1) NOT NULL,
    [Fullname] [nvarchar](255)
  COLLATE Latin1_General_CI_AS NULL,
    [PropertyName] [nvarchar](50)
  COLLATE Latin1_General_CI_AS NULL,
    [PropertyValue] [nvarchar](255)
  COLLATE Latin1_General_CI_AS NULL,
    [Attributes] [nvarchar](1000)
  COLLATE Latin1_General_CI_AS NULL,
 CONSTRAINT [PK_tProperty] PRIMARY KEY CLUSTERED
(
    [ID] ASC
)WITH (IGNORE_DUP_KEY = OFF)
)
END
GO
/****** Object:  Table [dbo].[tPerson]
 * Script Date: 01/30/2007 11:24:15 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
IF NOT EXISTS (SELECT * FROM sys.objects
WHERE object_id = OBJECT_ID(N'[dbo].[tPerson]')
AND type in (N'U'))
BEGIN
CREATE TABLE [dbo].[tPerson](
    [ID] [int] IDENTITY(1,1) NOT NULL,
    [Firstname] [nvarchar](50) COLLATE Latin1_General_CI_AS NULL,
    [Lastname] [nvarchar](50) COLLATE Latin1_General_CI_AS NULL,
 CONSTRAINT [PK_tPerson] PRIMARY KEY CLUSTERED
(
    [ID] ASC
)WITH (IGNORE_DUP_KEY = OFF)
)
END
GO
SET IDENTITY_INSERT [dbo].[tPerson] ON
INSERT [dbo].[tPerson] ([ID], [Firstname], [Lastname])
VALUES (1, N'Norbert', N'Eder')
INSERT [dbo].[tPerson] ([ID], [Firstname], [Lastname])
VALUES (2, N'Karoline', N'Draxler')
SET IDENTITY_INSERT [dbo].[tPerson] OFF
```

Download [Microsoft SQL Server Database Publishing Wizard 1.0]()

5.17.Kostenlose Code Coverage Tools

Was ist Code Coverage?

Die Code Coverage Analyse wird zusammen mit Unit Tests eingesetzt. Durch sie wird überprüft, welcher Anteil des Sourcecodes durch Tests abgedeckt wird. Dadurch kann weiters festgestellt werden, welche Bereiche nicht abgedeckt sind und hat so die Möglichkeit, dafür entsprechende Unit Tests zu bilden. Insgesamt wird durch den Einsatz der Code Coverage Analyse die Qualität der Unit Tests verbessert.

Kostenlose Code Coverage Tools für .NET?

Kommerzielle Produkte zu diesem Thema gibt es einige, jedoch auch kostenlose Tools sind verfügbar.

NCover ist eines dieser kostenlosen Tools und steht aktuell in der Version 1.5.5 beta zur Verfügung, welche jedoch nur mit dem .NET Framework 2.0 zusammenarbeitet und nicht abwärtskompatibel ist. Dazu muss zu einer älteren Version gegriffen werden. Für eine komfortablere Auswertung der Berichte bietet sich der kostenlose NCoverExplorer an.

Eine weitere Variante stellt PartCover dar. PartCover unterscheidet sich in einigen Bereichen von NCover, leistet jedoch auch gute Dienste.

Lohnt sich der Einsatz?

Egal ob nun ein kostenloses Code Coverage Tool oder ein kommerzielles eingesetzt wird, insgesamt wird die Qualität der Unit Tests und somit der gesamten Software erhöht. Daher: Sehr empfehlenswert.

5.18.C# - VB.NET Code Converter

Todd Anglin hat zusammen mit Telerik einen C# - VB.NET / VB.NET - C# Code Converter entwickelt. Neben einer bereits in der Beta befindlichen Online-Version und ein Widget für Yahoo!, soll es in naher Zukunft ebenfalls eine Version für den Google Desktop und die Windows Sidebar geben.

Mehr Informationen gibt es unter http://telerikwatch.com/2007/03/announcing-new-code-converter.html

Nachtrag:
Anscheinend verwendet der Code Converter nichts anderes als die SharpDevelop Parser Bibliotheken, wie man im Abschnitt *About* nachlesen kann. Das Original findet man also hier, neben der bereits erhältlichen Offline Variante.

5.19.IE7pro - Internet Explorer 7 Add-on

Das IE70pro Add-on erweitert den Internet Explorer 7 um ein paar nützliche Funktionen. Nach der Installation ist es möglich z.B. möglich mit Mausgesten zu arbeiten, oder nach einem Absturz alle geöffneten Tabs wiederherzustellen. Außerdem kann eine komplette Seite direkt als Bild abgespeichert werden. IE70pro ist kostenlos und kann unter folgender URL bezogen werden http://www.ie7pro.com/. Dort gibt es natürlich auch die komplette Featurelist und weitere Informationen.

5.20.Source Code Line Counter

In einem früheren Beitrag habe ich bereits ein Visual Studio Add-In vorgestellt, welches die Anzahl der geschriebenen Code-Zeilen auswertet. Das Programm Source Code Line Counter kommt jedoch nicht als Add-In, sondern als selbstständiges Programm daher. Dafür sind die Auswertungen nicht so umfangreich, aber dennoch brauchbar.

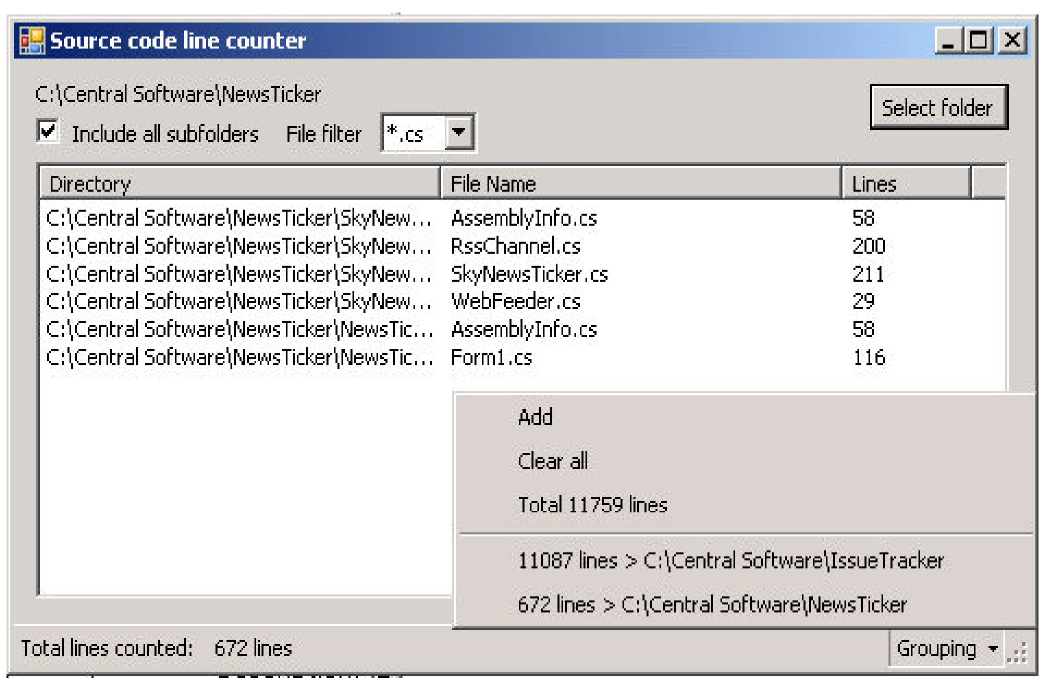


Abbildung 83: Source code line counter

Neben mehreren Option wie **Include all subfolders**, **Include blank lines** oder **Include auto generated code**, gibt es die Möglichkeit über den Befehl **Grouping** zwei unterschiedliche Projekte zusammenzuführen, und somit ein totales Ergebnis zu erhalten.

Das Programm selbst ist kostenlos und kann bei Bedarf erweitert werden, da der Quellcode ebenfalls verfügbar ist.

Mehr Infos unter http://www.codeproject.com/cs/files/SourceCodeLineCounter.asp

5.21..NET Reflector Add-Ins

Auf www.codeplex.com finden sich zahlreiche Add-Ins für den .NET Reflector von Lutz Roeder. .NET Reflector ist ein Class-Browser, Explorer, Analayzer und Documentation Viewer für .NET. Außerdem erlaubt er das Durchsuchen und Dekompilieren von .NET Assemblies.

http://www.codeplex.com/reflectoraddins

Der .NET Reflector gehört ebenfalls zu den Tools, die im .NET Casts - Podcast vom 04.02.2007 mit dem Thema *.NET Developement Tools* vorgestellt wurden.

5.22.UML Modellierung mit Visual Studio

Wer auf der Suche nach einem UML Modelling-Tool für Visual Studio ist, dem sei eine weitere Variante durch tangible engineering geboten. Zusätzlich zum kostenpflichtigen Tool tangible architect 4.0 gibt es auch eine kostenlose Variante die folgende Diagramme unterstützt:

- Case Diagrams
- Component Diagrams
- State Charts
- Class Diagrams
- Activity Diagrams
- Persistent Object Models

Hier ein Screenshot:

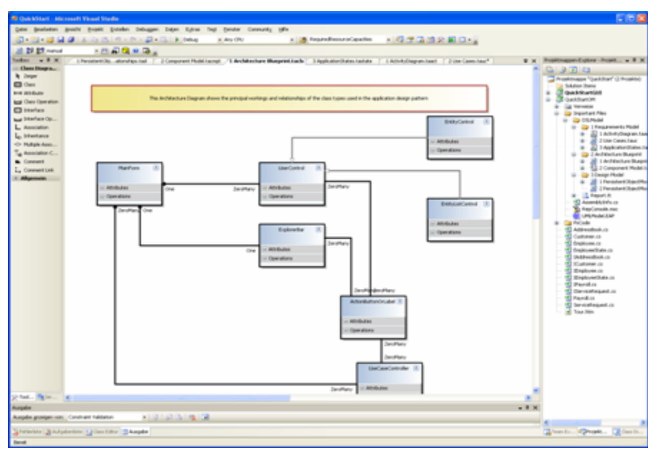


Abbildung 84: tangible engineering: UML Modellierungstool

Die erstellten Diagramme und Modelle sind mit der kostenpflichtigen Variante kompatibel.

5.23.Tool: XmlExplorer

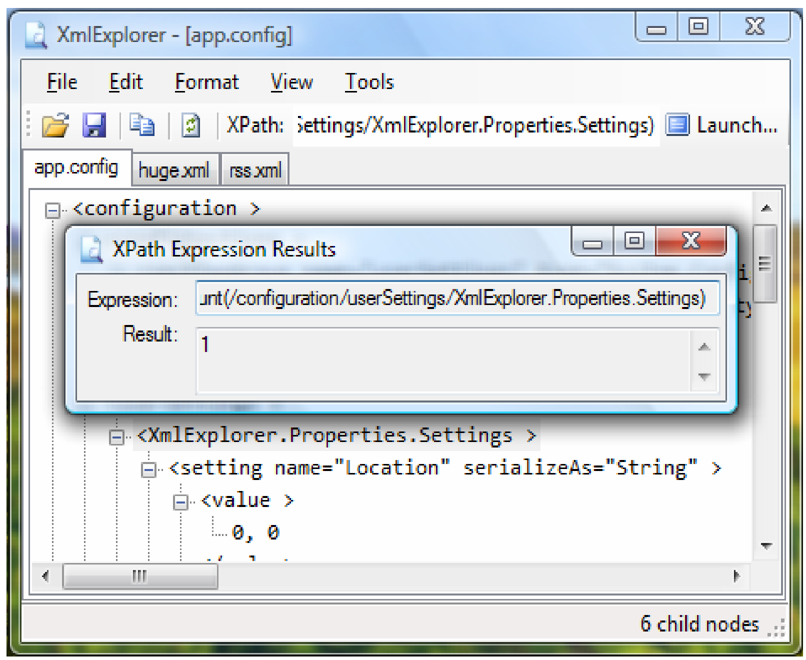


Abbildung 85: XmlExplorer

Wer beispielsweise das XML Notepad verwendet, der sollte zusätzlich einen Blick auf den XML Explorer werfen. Vor allem im Vergleich zum XML Notepad kann dieses Tool mit einer wesentlich besseren Performance, als auch mit weniger Speicherverbrauch aufwarten. Allerdings fehlt die Editierfunktion. Es ist jedoch wunderbar geeignet, eine XML Datei in einer Baumstruktur anzeigen zu lassen. Zudem sind einige hilfreiche Funktionen inkludiert. Einfach mal ansehen.

5.24.NClass - kostenloser UML Klassen Designer

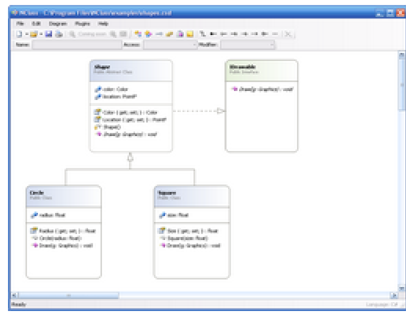


Abbildung 86: NClass

Wer einen leichtgewichtigen UML Klassen Designer sucht, der findet ein kostenloses Tool in NClass. Zwar bietet es nicht allzu viele Funktionalitäten, dennoch sollte es genügen, um einen schnellen Überblick über ein kleines Projekt zu erhalten. Für größere Projekte eignet sich das

Tool sicherlich nicht, da dafür entsprechende Funktionen einfach fehlen. Da das Projekt jedoch erst im Oktober 2006 erstmalig veröffentlicht wurde, darf noch einiges zu erhoffen sein.

Aktuelle Features

- Precision snapping to align objects without grid
- Support for two languages: C# and Java
- Declaration parser for class members to edit them faster
- Strict syntactical/semantical controlling
- Configurable diagram styles
- Multilanguage user interface
- Printing/saving to image

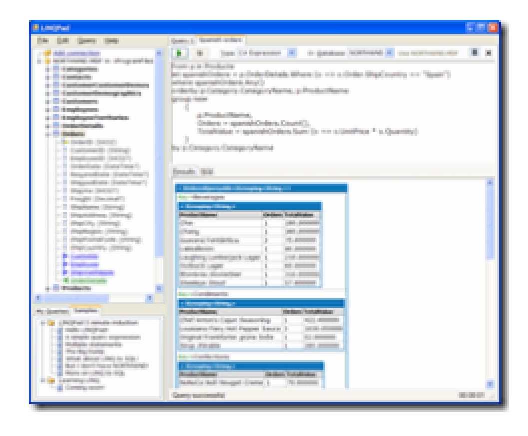
Geplante Features

- Association name, role and cardinality fields
- Zooming in/out
- Packages
- Class and member comments for documenting
- Source generator
- Disassemble .NET assemblies

Es lohnt sich auf jeden Fall, das Projekt im Auge zu behalten.

5.25.LINQPad - LINQ Ausdrücke testen

Joseph Albahari hat ein Tool geschrieben mit dem es auf einfache und schnelle Weise möglich ist LINQ Ausdrücke zu testen - ähnlich dem SQL Query Analyzer. Das Tool unterstützt sowohl LINQ to Objects, als auch LINQ to SQL und LINQ to XML. Nach dem Download kann das Tool direkt ohne Installation gestartet werden, vorausgesetzt das .NET Framework 3.5 Beta 2 ist installiert.



Mehr Infos, sowie den Download gibt es hier.

6. Microsoft Office

6.1. Word 2007: Custom Ribbons erstellen

Im Beitrag Word 2007 Winamp Controller habe ich bereits ein Beispiel für Custom Panes gezeigt.

Dieser Beitrag zeigt nun, wie einfach ein Custom Ribbon (Ribbons werden die neuen Menüleisten unter Office 2007 genannt) erstellt werden kann. Für den Moment muss man sich mit dem Sourcecode zufrieden geben. Entsprechende Tutorials sind in Arbeit und werden demnächst veröffentlicht.

Soviel sei verraten: Ein wenig Windows Presentation Foundation und schon kann es losgehen.

Das Endergebnis des Beispiels [1] sieht folgendermaßen aus:

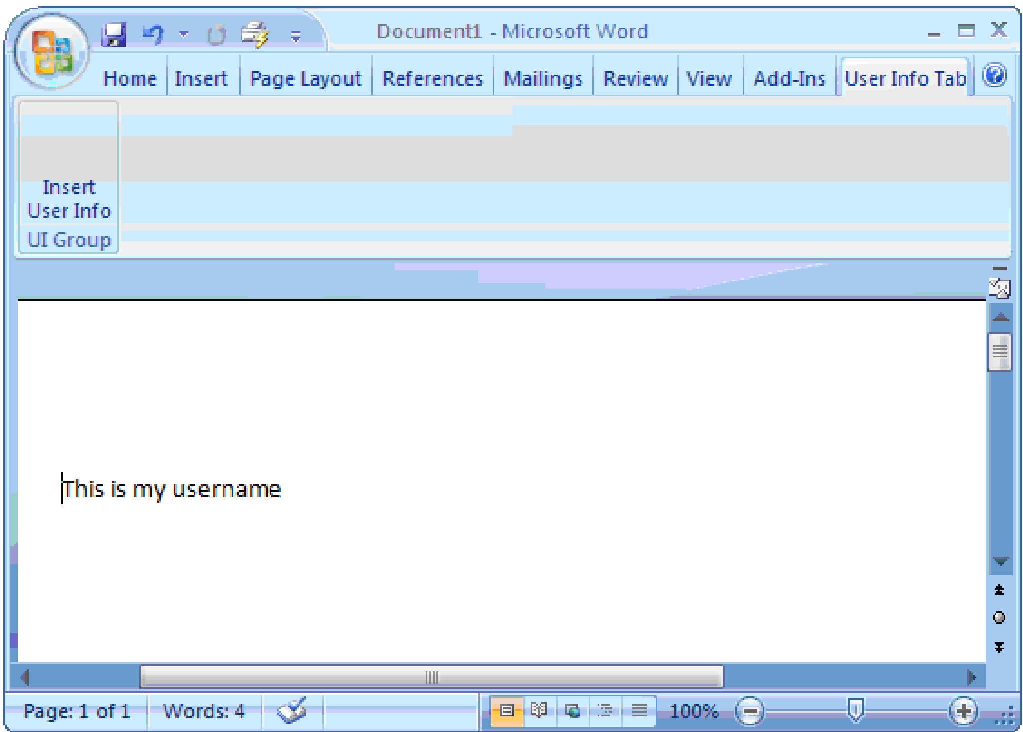


Abbildung 87: Word 2007 - Custom Ribbons erstellen

Das Beispiel liegt in einer Visual Studio 2005 Solution vor und wurde in C# erstellt.

Zu diesem Thema steht bereits ein Tutorial [2] zur Verfügung.

[1] Download Word 2007 Custom Ribbon Beispiel
[2] Zur Tutorials-Page

6.2.MS Outlook - Makros reloaded

So, nachdem es gestern den "Mail mittels Shortcut als unread markieren"-Tag gegeben hat, hab ich noch ein kleines Makro gebaut, welches mir mittels Shortcut den "Junk E-Mail"-Folder und auch gleich den "Gelöschte Objekte"-Order leert. Damit ist's per Tastendruck sauber im Outlook.

Hier nun das Makro:

```
Sub RemoveJunkAndDeleted()

Dim mItem As MailItem
Dim mNamespace As NameSpace
Dim junkFolder As MAPIFolder
Dim deletedFolder As MAPIFolder

Set mNamespace = Application.GetNamespace("MAPI")

Set junkFolder = mNamespace.GetDefaultFolder(olFolderJunk)


For Each mItem In junkFolder.Items
mItem.Delete
Next

Set deletedFolder = mNamespace.GetDefaultFolder(olFolderDeletedItems)

For Each mItem In deletedFolder.Items
mItem.Delete
Next

End Sub
```

Dieses Makro ist wieder wie gestern im Beitrag [1] einzubinden. Als Shortcut habe ich ein k vergeben, da dies noch nicht benutzt wird.

PS: Den Junk-E-Mail-Folder immer schon sauber halten. Denn Outlook archiviert diesen Ordner standardmäßig mit, was ich zwar nicht verstehe, ist aber so. Einfach die Folder-Eigenschaften öffnen und die Archivierung deaktivieren.

[1] MS Outlook: Mails mit Shortcut als gelesen markieren

6.3.MS Outlook - Makros reloaded 2

Vor einiger Zeit habe ich ein kleines VBA-Makro veröffentlicht, mit dem es einfach möglich ist, die Ordner "Junk E-Mail" und "Gelöschte Objekte" zu leeren (siehe MS Outlook - Makros reloaded).

Alex Bierhaus hat mich nun auf einen Fehler dieses Makros hingewiesen, den ich in der Hitze des Gefechts übersehen hatte:

In den gelöschten Objekten befinden sich natürlich nicht nur gelöschte Emails, sondern auch gelöschte Kontakte, Notizen, Aufgaben etc. Hierbei wurde ein Fehler ausgelöst und die Objekte wurden nicht gelöscht. Hier nun eine marginal geänderte Variante, die nun einwandfrei funktionieren sollte.

```
Sub RemoveJunkAndDeleted()

Dim mItem As Object
Dim mNamespace As NameSpace
Dim junkFolder As MAPIFolder
Dim deletedFolder As MAPIFolder

Set mNamespace = Application.GetNamespace("MAPI")

Set junkFolder = mNamespace.GetDefaultFolder(olFolderJunk)


For Each mItem In junkFolder.Items
mItem.Delete
Next

Set deletedFolder = mNamespace.GetDefaultFolder(olFolderDeletedItems)

For Each mItem In deletedFolder.Items
mItem.Delete
Next

End Sub
```

Getestet wurde mit gelöschten Emails, Kontakten, Aufgaben und Notizen unter Verwendung von Microsoft Office 2003.

6.4.MS Outlook: Mails mit Shortcut als gelesen markieren

Das 'gelesen markieren' funktioniert unter Outlook nur, wenn man zuvor die entsprechenden Mails markiert und dann STRG-Q drückt. Manchmal möchte man jedoch alle Mails in einem ausgewählten Folder als gelesen markieren - und zwar mit Hilfe eines Shortcuts.

Hier gibt es die Info wie das funktioniert.

Zuerst ist folgendes Makro anzulegen:

```
Sub SetAllRead()
Dim mItem As MailItem
If Outlook.ActiveExplorer.CurrentFolder.UnReadItemCount > 0 Then
For Each mItem In Outlook.ActiveExplorer.CurrentFolder.Items
mItem.UnRead = False
Next
End If
End Sub
```

In meinem Beispiel nennt sich das Makro `SetAllRead`. Nach dem Erstellen ist das Menü **Extras/Anpassen** zu wählen. In der Lasche **Befehle** nun einfach den Eintrag **Makros** in der linken Liste auswählen. Das gerade erstellte Makro sollte in der

rechten Liste verfügbar sein. Mit der linken Maustaste einfach das Makro anfassen und in eine Toolbar ziehen.

Jetzt kommt der entscheidende Teil bezüglich Shortcut: Mit einer rechten Maustaste, auf die eben erstellte Schaltfläche (Achtung, das Anpassen-Fenster muss noch geöffnet sein) erscheint ein Menü, in dem ein Name für die Schaltfläche vergeben werden kann. Hier ist ein Name zu wählen, in dem ein Buchstabe mit einem kaufmännischen Und (&) hervorzuheben ist. Dieser wird dann als Shortcut verwendet. Hier sollte ein Buchstabe gewählt werden, für den noch kein Shortcut vergeben ist. Ich denke, ein m würde sich hier anbieten.

Ab sofort kann man einzelne Mailfolder auswählen, ALT+M drücken und schon werden die darin enthaltenen MailItems als gelesen markiert.

7. Datenbank-Management-Systeme

7.1. SQL Server 2000 Replikation und Error 18483

Die Replikation lässt sich nicht einrichten, da der Benutzer **distributor_admin** nicht als Remotebenutzername eingetragen ist. Und wo genau liegt nun das Problem? Die Antwort ist eigentlich ganz einfach:

Das Problem liegt daran, dass der Servername des SQL Servers nicht mit der eingetragenen `ServerName`-Property überein stimmt. Vermutlich durch eine Umbenennung etc. Diese Einstellungen kann man mit folgender Abfrage herausfinden:

```
SELECT @@SERVERNAME, SERVERPROPERTY('ServerName')
```

Die beiden Felder des Resultates müssen hier gleich sein. Wenn nicht, dann ist folgendes Script auszuführen:

```
USE master
GO

DECLARE @serverproperty_servername varchar(100),
@servername varchar(100)

SELECT @serverproperty_servername = CONVERT(varchar(100),
SERVERPROPERTY('ServerName'))

SELECT @servername = CONVERT(varchar(100), @@SERVERNAME)

EXEC sp_dropserver @server=@servername, @droplogins='droplogins'

EXEC sp_addserver @server=@serverproperty_servername, @local='local'
```

Damit wird die Serverregistrierung gelöscht und neu gesetzt, mit dem Wert aus dem Feld `SERVERPROPERTY('ServerName')`. Wird ein Server umbenannt, betrifft dies nur diese Eigenschaft, der eigentliche Servername bleibt jedoch davon unberührt. Weiters sollte die Stored Procedure `sp_dropserver` mit `@droplogins` aufgerufen werden, um etwaige vorhandene Remotelogins ebenfalls zu löschen. Andernfalls kann der Server nicht neu registriert werden.

Nach diesen Schritten ist der SQL Server neu zu starten und es kann nun versucht werden, die Replikation zu konfigurieren. Nun sollte es funktionieren.

Der ursprüngliche Tipp ist auf der Microsoft-Support-Seite [1] zu finden, jedoch ohne den `@droplogins` Hinweis.

[1] Replication setup is not successful when SQL Server is deployed by using a disk image

7.2.SQL Server 2005: Output-Klausel

Eines der neuen Features des SQL Server 2005 ist die `OUTPUT`-Klausel, die Transact-SQL um eine nützliche Funktion erweitert.

Durch die Angabe der OUTPUT-Klausel können in `INSERT`, `UPDATE` und `DELETE` Anweisungen, die betroffenen Datensätze, oder Teile davon (beispielsweise die Primary-Key-Werte) zurückgegeben werden.

Dies ist beispielsweise bei einem `INSERT` sehr interessant, wenn die vergebene ID im weiteren Ablauf benötigt wurde. Diese konnte bisher via `SELECT @@Identity` bezogen werden. Danach musste ein weiteres `SELECT`-Statement abgesetzt werden, um den Datensatz zu erhalten. Durch die `OUTPUT`-Klausel ist dies nicht mehr notwendig. Die Ergebnisse werden sofort zurückgeliefert, sofern gewünscht.

Beispiel

```
USE AdventureWorks;
GO
DELETE tUser
OUTPUT deleted.*
WHERE ID = 7;
GO
```

Dieses Beispiel löscht den Datensatz mit der ID 7 aus der Tabelle `tUser` und gibt den gesamten Datensatz zurück. Es können jedoch auch nur einzelne Felder zurückgeliefert werden:

```
USE AdventureWorks;
GO
DELETE tUser
OUTPUT deleted.ID, deleted.Firstname, deleted.Lastname
WHERE ID = 7;
GO
```

Hier würden nur die ID, der Vorname und der Nachname als `ResultSet` zurückgegeben werden.

Wie kann dies unter dem .NET Framework benutzt werden? Es verhält sich sehr einfach. Die einzelnen Statements werden mit einem Command-Objekt (beispielsweise `SqlCommand`) abgesetzt. Im Falle von `INSERT`, `UPDATE` bzw. `DELETE` Abfragen wird dazu die Methode ExecuteNonQuery verwendet. Wird nun die `OUTPUT`-Klausel verwendet, ist anstatt der Methode `ExecuteNonQuery` beispielsweise die Methode `ExecuteReader` zu verwenden. Mit dem zurückgegebenen `DataReader` kann durch die einzelnen Datensätze iteriert werden.

Referenzen

[1] MSDN

7.3.Objekte und relationale Daten in einer Datenbank

Prinzipiell gibt es den Ansatz der ORDBMS (Objektrelationale Datenbank Management Systeme). Dabei wird ein relationales System um objektorientierte Fähigkeiten erweitert. Oft wird auch nur ein objektorientierte Zugriffsschicht darüberlegt. In der Tat werden allerdings keine Objekte mit relationalen Daten vermischt.

Wie würde das Speichern von Objekten zusammen mit relationalen Daten in einer Datenbank in der Praxis aussehen?

Prinzipiell stehe ich dem Vermischen von Objekten mit relationalen Daten eher negativ gegenüber. Der Ansatz von XML-Feldern in einer Datenbank (siehe SQL Server 2005) ist durchaus praktisch und in manchen Fällen auch sinnvoll. Dies jedoch zu nutzen, um Objekte abzulegen macht eher weniger Sinn. Aus folgenden Gründen (wenn ich von Serialisierung spreche, beziehe ich mich auf die XML-Serialisierung):

- Aufwand der Serialisierung. Bedingt durch diesen Aufwand können die Daten ohnehin eder in eine relationale Struktur gequetscht werden. Performancemässig wird es hier (allerdings hab ich das jetzt nicht getestet) nicht sehr viel Unterschied geben.
- Weiters verleitet dieser Ansatz dazu, eine Tabelle mit mehreren XML-Feldern zu erstellen und darin serialisierte Objekte abzulegen. Eventuell noch von unterschiedlichen Typen. Spätestens dieser Punkt würde durchaus Probleme aufwerfen.
- Durch das einfache serialisierte Ablegen würden in der Datenbank Referenzen nicht mehr ersichtlich sein. In einem reinen relationalen oder reinen objektorientierten System bleibt dieser Vorteil erhalten. Referenzen in XML sind zwar möglich, aber selbst bei einem eigenen XML-Serializer nützt das Einfügen dieser Referenzen wenig, wenn sie durch die Datenbank nicht dargestellt werden können.

Abfragen sind natürlich auch in dieser Form noch auf der Datenbank möglich ohne alle Daten zu laden, aber durch die Vermischung von SQL und XPATH etc. würde ich meinen, dass der Performance-Vorteil eher gering ausfällt -> dürfte wohl im Minusbereich angesiedelt sein.

Und zu guter Letzt: Diese Variante würde kein gemeinsames Ablegen von Objekten und relationalen Daten implizieren. Das einzige was Sinn machen würde, wäre das Ablegen von relationalen Informationen (Metadaten etc.) zu Objekten - aber ich denke das übernimmt ohnehin die Indizierung für uns.

7.4. SQL Server 2005: Erstellung einer Replikation

Wer eine Replikation unter dem SQL Server 2005 erstellen muss bzw. will, der findet im nachfolgenden Tutorial eine Anleitung. Alle notwendigen Schritte werden anhand von Screenshots und Beschreibungen aufgezeigt und sollten recht einfach nach zu vollziehen sein.

Datenreplizierung unter dem SQL Server 2005 (1.1 MB)

7.5. SQL Server 2005: Managed Code

Dieses Tutorial soll zeigen, wie managed Code (in diesem Fall wird C# benutzt) unter dem SQL Server 2005 verwendet werden kann.

Managed Code unter dem SQL Server 2005 (213 KB)

7.6. SQL Server 2005: Custom Datatypes

Eine Einführung in die benutzerdefinierten Datentypen unter dem SQL Server 2005 bietet das unten verlinkte Dokument.

Datentypen unter dem SQL Server 2005 (338 KB)

7.7. C# und SQL Server 2005

Immer wieder stoße ich auf Anfragen, wie denn genau eine Anbindung an den SQL Server 2005 mit C# (oder VB.NET) funktioniert. Grundsätzlich gibt es dazu zahlreiche Tutorials im Internet, die meisten jedoch auf Englisch. Diese Tatsache scheint dann doch sehr viele abzuschrecken. Daher habe ich mich entschlossen, ein Tutorial zu diesem Thema zu verfassen, um eine entsprechende deutschsprachige Ressource bereitstellen zu können.

Inhalt

1. Einführung
2. Verbindung herstellen
3. Daten abfragen
4. Daten hinzufügen
5. Zusammenfassung

1. Einführung

Für den Zugriff auf den SQL Server (2000 oder 2005 ist hierbei unerheblich) werden vom .NET Framework alle notwendigen Funktionen zur Verfügung gestellt. Diese verstecken sich im Namespace `System.Data.SqlClient`. Für den SQL Server könnten zwar auch die Klassen aus dem Namespace `System.Data.OleDb` verwendet werden, jedoch ist der `DataProvider` unter `SqlClient` auf den SQL

Server optimiert und sollte (außer andere triftige Gründe sprechen dagegen) verwendet werden.

2. Verbindung herstellen

Nun, starten wir damit, eine Verbindung zum SQL Server herzustellen. Dazu verwenden wir die Klasse `SqlConnection` aus dem oben angegebenen Namespace. Der KLasse `SqlConnection` muss ein `ConnectionString` übergeben werden. Dieser definiert wo der SQL Server zu finden ist, welche Datenbank verwendet werden soll und wie die User-Informationen lauten. Beispielsweise würde ein ConnectionString wie folgt aussehen:

```
Data Source=Aron1;Initial Catalog=pubs;User Id=sa;Password=asdasd;
```

Die Data Source stellt den Computer dar, auf dem der SQL Server läuft. Zu achten ist hierbei, dass der SQL Server 2005 mit einem Instanznamen angelegt wird, welcher im ConnectionString entsprechend anzugeben ist, da eine Verbindung sonst nicht aufgebaut werden kann. Die Einstellung Initial Catalog beschreibt den Namen der Datenbank auf welche verbunden werden soll. User Id ist der zu verwendende Username und Password sollte selbst sprechend sein. Weitere Informationen zu den ConnectionStrings sind unter [1] zu finden.

In C# sieht ein Verbindungsaufbau nun wie folgt aus:

```
SqlConnection conn = new SqlConnection(@"Data
Source=COMPUTERNAMESQLEXPRESS;Initial Catalog=DatabaseDemo;User
Id=sa;Password=MyPassword;");
conn.Open();

conn.Close();
```

In diesem Fall ist ein SQL Server 2005 Express installiert. `COMPUTERNAME` stellt den Namen des Computers dar und SQLEXPRESS ist der Instanzname des SQL Servers. Dieser Name wird im Normalfall bei der Installation eingegeben. Ist nicht sicher wie dieser lautet, dann kann dies in der Dienste-Liste ausgelesen werden. Dazu einfach die Dienste anzeigen lassen, den SQL Server Dienst suchen, in Klammer dahinter steht der entsprechende Instanzname.

Durch den obigen C# Code kann man sich nun zur Datenbank verbinden. Für das Öffnen der Verbindung ist die Methode `Open()` zu verwenden. `Close()` sorgt für das Schließen der Verbindung.

3. Daten abfragen

Da wir nun eine Verbindung aufbauen können, erfolgt der nächste Schritt: Daten abzufragen. Hierfür werden die Klassen `SqlCommand` und `SqlDataReader` verwendet.

Folgender Code übernimmt das für uns:

```
SqlCommand com = new SqlCommand("SELECT * FROM tPerson", conn);

SqlDataReader reader = com.ExecuteReader();
while (reader.Read())
```

```
{
Console.WriteLine("ID: {0}, Firstname: {1}, Lastname: {2}", reader[0].ToString(),
reader[1].ToString(), reader[2].ToString());
}
reader.Close();
```

Hier passiert nun folgendes: Zuerst erstellen wir einen `SqlCommand`, der auf der Datenbank ausgeführt werden soll. Als Parameter übergeben wir ein SQL-Statement und die geöffnete Verbindung `conn`.

Danach instanzieren wir einen `SqlDataReader`. Dieser ist für das Auslesen der Daten zuständig. Der Befehl `com.ExecuteReader()` sorgt dafür, dass der Command ausgeführt wird und ein `DataReader` zurückgegeben wird. Mit dem `DataReader` kann Datensatz für Datensatz durch das Ergebnis iteriert werden, was wir auch mit Hilfe der `while`-Schleife tun. Nachdem die Daten bezogen wurden, ist der `DataReader` wieder zu schließen. Darin geben wir die Daten lediglich auf der Console aus. Die Ausgabe würde wie folgt aussehen:

```
ID: 1, Firstname: Norbert, Lastname: Eder
ID: 2, Firstname: Test, Lastname: Person
ID: 3, Firstname: Franz, Lastname: Nachname
```

Es empfiehlt sich, auch die weiteren Möglichkeiten der Klasse `SqlCommand` im MSDN [2] nachzuschlagen, da viele weitere Funktionen zur Verfügung stehen. Alle zu beschreiben würde jedoch den Umfang dieses Tutorials sprengen.

4. Daten hinzufügen

Da wir nun eine der Möglichkeiten gesehen haben wie Daten abgefragt werden können, werden wir nun einen Datensatz hinzufügen. Dies passiert wie folgt:

```
SqlCommand com = new SqlCommand("INSERT INTO tPerson (Firstname, Lastname) VALUES
(@firstname, @lastname)", conn);
com.Parameters.Add(new SqlParameter("@firstname", "fritz"));
com.Parameters.Add(new SqlParameter("@lastname", "huber"));
com.ExecuteNonQuery();
```

Wieder erstellen wir einen `SqlCommand`, der ein SQL Statement übergeben bekommt sowie die Datenbankverbindung. Hier passiert dann noch etwas Spezielles. Aus Sicherheitsgründen werden die Daten nicht direkt ins SQL Statement geschrieben, sondern via `SqlParameter`. Dieser Schritt hat einige Vorteile was das Thema **SQL Injection** etc. betrifft. Sollte also immer in dieser Form angewandt werden. Mittels der `ExecuteNonQuery`-Methode des `SqlCommand`-Objektes wird der Befehl auf der Datenbank ausgeführt und der Datensatz eingefügt.

4. Zusammenfassung

Dies war ein kurzer Einblick in die Datenbank-Thematik unter C#. Alle Klassen und Methoden können auf die gleiche Art und Weise unter VB.NET verwendet werden. Es empfiehlt sich, die angegebenen Klassen im MSDN genauer anzusehen, sowie auch die vorhandenen Beispiele zu den einzelnen Methoden durch zuarbeiten. Danach sollte dieses Thema kein Problem mehr darstellen.

[1] http://www.connectionstrings.com
[2] http://msdn2.microsoft.com oder http://msdn.microsoft.com

7.8. OODBMS- Object Oriented DataBase Management Systems

Objektorientierte Datenbanken sind schon sehr nett, aber leider auch nicht für alles einsetzbar. Das schöne daran ist, dass die Objekte an sich gespeichert werden, nicht ein relationales Mapping davon. Dies bedeutet, dass das Abbilden von relationalen Daten auf das ursprüngliche Objekt nicht mehr notwendig ist.

Allerdings sehe ich hier doch noch einige Probleme, die nicht unausgesprochen bleiben sollten:

1. Fehlende Standards

Einheitliche Standards zu ORDBMS bzw. OODBMS gibt es nicht - auch wenn es diese Systeme nicht erst seit gestern gibt. Dadurch ist es nicht gewährleistet, dass alle Systeme auf gleiche Art und Weise arbeiten. Object-oriented SQL (OSQL) ist ansich recht nett, aber auch hier gibts die gleichen "Fehler" wie beim herkömmlichen SQL -> Jeder implementiert eine eigene Variante.

2. Backup, Restore, Verteilung etc.

Objekte sind wesentlich komplexer als relationale Strukturen. Entsprechend komplexer wird dadurch auch ein Backup, ein Restore, eine Verteilung oder überhaupt nur die Aktualisierung der Daten, das Nachziehen von Referenzen usw. Diese Dinge sind mit hohem Aufwand verbunden, daher ist eine OODBMS in diesen Gebieten sicherlich langsamer als ein herkömmliches RDBMS.

3. Angst?

Das Thema OODBMS ist an sich genau so alt, wie es objektorientierte Sprachen sind. Letztere haben sich in einigen Bereichen (Enterprise etc.) durchgesetzt. Erstere nicht. Warum? Nun, relationale Datenbanken sind einfacher zu handhaben und wurden über lange Zeit erprobt. Viele haben davor Angst, auf eine "neue" Technologie zu setzen (Anm.: die jedoch nicht viel jünger ist). Dadurch fließt auch nicht soviel Entwicklungsarbeit in objekt-orientierte Ansätze bei Datenbank-Management-Systemen.

4. Fehlende Unterstützung?

Viele Lösungen entstehen und verschwinden entsprechend auch wieder, da die Nutzung nicht vorhanden ist. Ich persönlich kenne nur ein OODBMS welches sich lange gehalten hat. Caché [1]. Alle anderen sind entweder "gemeinnützige" Projekte mit mehr oder weniger geringem Support.

Natürlich gibt es da noch andere Punkte. Der Vollständigkeit halber stehen dem jedoch auch sehr viele Vorteile gegenüber - keine Frage. Allerdings sehe ich hier db4o [2] nicht unbedingt das als Gelbe vom Ei an. Aber der Weg den db4o beschreitet ist prinzipiell nicht so schlecht.

[1] http://www.intersystems.com/cache/
[2] http://www.db4o.com/

7.9.SA User unter SQLServer 2005 umbenennen

Unter dem Microsoft SQL Server 2000 ist es ja nicht möglich, den **sa** User umzubenennen bzw. zu deaktivieren. Der SQL Server 2005 bietet allerdings diese Möglichkeit. Durchgeführt können diese beiden Aktionen mit **ALTER LOGIN** werden.

```
ALTER LOGIN sa DISABLE;
ALTER LOGIN sa WITH NAME = [sys-admin];
```

7.10.SQL Server 2000: Felder zur Replikation hinzufügen

In manchen Fällen ist es notwendig, zu einer Datenbank neue Felder hinzuzufügen. Wird nun diese Datenbank auf einen weiteren Server repliziert ergeben sich hier die einen oder anderen Probleme.

Die beste Erfahrung habe ich mit der Stored Procedure `sp_repladdcolumn` gemacht. Mit Hilfe dieser Stored Procedure wird das Feld in der zu replizierenden Datenbank (Publication) eingetragen und zur Replikation hinzugefügt.

Wichtig hierbei ist, dass die Replikation vor der Änderung gestoppt werden muss.

```
exec sp_repladdcolumn
@source_object = 'Tabellenname',
@column = 'Spaltenname',
@typetext = 'float NULL',
@publication_to_add = 'all'
```

Danach muss der Snapshot erneut erstellt und nach Abschluss dessen die Replizierung wieder gestartet werden.

Mit der Stored Procedure `sp_addarticle` kann übrigens eine neue Tabelle erstellt werden, die ebenfalls zur Replizierung hinzugefügt wird.

7.11.SQL Server 2000: Einschränkungen bei Replikation

Bei der Replikation unter dem Microsoft SQL Server 2000 gibt es eine vorkonfigurierte Einschränkung bei `text`-, `ntext`- und `image`-Feldern. Die Feldgröße bei der Replizierung ist auf 65536 Zeichen beschränkt. Wird nun in ein Feld ein größerer Wert eingefügt, kommt es zum folgenden Fehler:

Englisch:
Length of text, ntext, or image data (x) to be replicated exceeds configured maximum 65536.

Deutsch:
Länge der zu replizierenden text-, ntext- oder image-Daten übersteigt das konfigurierte Maximum 65536.

Dies kann mit folgender Stored Procedure umgangen werden:

```
CREATE PROC usp_configure_maxReplSize

   @NewSize int = 1000000

AS

    exec sp_configure 'max text repl size'
    exec sp_configure 'max text repl size', @NewSize

    RECONFIGURE WITH OVERRIDE

    exec sp_configure 'max text repl size'

GO
```

In diesem Fall wird die maximale Größe auf eine Million Zeichen gestellt.

7.12.SQL Server Replizierung: Alt oder doch neu?

Wer sich derzeit mit den Gedanken über eine SQL Server Replizierung spielt und sich nicht sicher ist, ob er den altbewährten SQL Server 2000 oder dann doch den neuen 2005er nehmen soll, dann kann ihm geholfen werden.

Werden in Zukunft Änderungen an Tabellen etc. vorgenommen?

Wenn nein, dann sehe ich kein Problem bei der Verwendung der 2000er-Version. Handelt es sich dabei allerdings um ein wachsendes System, d.h. die Struktur der Tabellen etc. kann bzw. wird sich verändern, dann rate ich eindeutig zum 2005er.

Der Grund liegt darin, dass Änderungen an der Tabellenstruktur im SQL Server 2000 nur schwer nachgezogen werden kann. Wird beispielsweise im Verteiler (der Server, der die Daten an seine Abonnenten verteilt) eine neue Tabelle angelegt, muss die Publikation, als auch die Abonnenten am Verteiler gelöscht und neu erstellt werden. Ändern sich nur einzelne Datenfelder, können diese nur mit der Stored Procedure `sp_repladdcolumn` zur Replikation hinzugefügt werden. Dies ist auf Dauer doch recht mühsam. Der SQL Server 2005 bietet diese Einschränkungen nicht mehr.

7.13.Autoinkrement-Wert nach INSERT INTO auslesen, die Zweite

Im letzten Beitrag zu diesem Thema zeigte ich kurz auf wie es möglich ist, einen inkrementellen Identitätswert nach einem INSERT auszulesen. Dazu verwendete ich die Konstante @@IDENTITY.

Allerdings ist die Verwendung dieser Konstante nicht immer zu empfehlen, da @@IDENTITY unabhängig des aktuellen Gültigkeitsbereiches arbeitet. Verwendet man z.B. einen Trigger, der nach dem eigentlichen INSERT ein weiteres auf eine andere Tabelle absetzt, erhält man den letzten Identitätswert. In diesem Fall also den Wert des zweiten INSERT.

Mit der Verwendung von SCOPE_IDENTITY() umgeht man dieses Problem, da diese Funktion in Abhängigkeit des aktuellen Gültigkeitsbereichs arbeitet.

```
CREATE PROCEDURE [sp_Test]
    (@name [varchar](300))
AS
    SET NOCOUNT ON
    INSERT INTO [Tabelle1]([name]) VALUES (@name)
    SELECT SCOPE_IDENTITY()
    SET NOCOUNT OFF
    INSERT INTO [Tabelle2] (Tabelle1_ID,[name]) VALUES (SCOPE_IDENTITY(),@name)
GO
```

Jedem, der mehr zum Thema SQL und SQL-Server wissen möchte, kann ich die Internetseite InsideSQL.de empfehlen.

7.14.Zufälligen Datensatz ausgeben

Möchte man einen zufälligen Datensatz aus einer Datenbank-Tabelle anzeigen, kann man dies aufwendig per C# Code lösen. Allerdings gibt es eine viel einfachere Methode direkt per SQL.

```
SELECT TOP 1 Feld FROM Tabelle ORDER BY NEWID()
```

Die Anweisung NEWID() generiert einen eindeutigen Wert vom Typ uniqueidentifier. In Verbindung mit einem ORDER BY erhält man nun einen "zufälligen" Datensatz.

Dieses Vorgehen funktioniert ebenfalls unter MySQL, allerdings mit einer etwas abgewandelten Syntax.

```
SELECT Feld FROM Tabelle ORDER BY RAND() LIMIT 1
```

7.15.Autoinkrement-Wert nach INSERT INTO auslesen

Häufig verwendet man in Tabellen einer Datenbank einen inkrementellen ID-Wert um Datensätze eindeutig zu identifizieren. Möchte man nun diesen ID Wert nach dem INSERT direkt weiterverwenden, drängt sich im ersten Moment die Lösung auf, direkt

nach dem INSERT ein SELECT auf die höchste ID abzusetzen und den Wert somit auszulesen.

Verwendet man diese Lösung, läuft man allerdings Gefahr eine falsche ID zu bekommen. Es ist durchaus möglich, dass zwischen dem INSERT und dem SELECT einer Session, bereits ein weiterer Datensatz einer anderen Session eingefügt wurde.

In diesem Beispiel verwende ich eine Stored Procedure um zwei Datensätze in unterschiedliche Tabellen einzufügen. Die erste Anweisung gibt einen inkrementellen Wert zurück, welcher in der zweiten Anweisung verwendet wird.

```
CREATE PROCEDURE [sp_Test]
    (@name [varchar](300))
AS
    SET NOCOUNT ON
    INSERT INTO [Tabelle1]([name]) VALUES (@name)
    SELECT @@IDENTITY
    SET NOCOUNT OFF
    INSERT INTO [Tabelle2] (Tabelle1_ID,[name]) VALUES (@@IDENTITY,@name)
GO
```

Zunächst wird per SET NOCOUNT ON angewiesen, dass die Anzahl der betroffenen Zeilen nicht als Teil des Ergebnises des INSERT INTO zurückgegeben wird. Per INSERT INTO Anweisung wird nun der Datensatz in die erste Tabelle eingefügt. Anschließend wird per SELECT @@IDENTITY der ID-Wert des eingefügten Datensatzes ausgelesen. Die Variable @@IDENTITY enthält immer den zuletzt eingefügten Identitätswert. Nun kann per zweiter INSERT INTO Anweisung der nächste Datensatz eingefügt werden.

Natürlich ist es möglich diese Lösung ohne Stored Procedure zu verwenden, um z.B. die ID direkt im Code weiter zu verwenden.

```
SET NOCOUNT ON;INSERT INTO [Tabelle1]([name]) VALUES (@name);SELECT @@IDENTITY AS
NewID
```

Die inkrementelle ID wird nun als Feld NewID ausgegeben.

7.16.Import einer CSV Datei mit Hilfe von BULK INSERT

BULK INERTS ermöglichen ein sehr schnelles Einfügen von vielen Daten mit einer Sql-Anweisung. Voraussetzung für diese Art Import ist allerdings der Microsoft SQL-Server. Des Weiteren muss sich die Import Datei ebenfalls auf dem SQL-Server befinden.
Für den Import bereiten wir zunächst die Tabelle in der Datenbank vor. Der Aufbau muss identisch mit der CSV Datei sein. In meinem Beispiel verwende ich die Spalten Daten1, Daten2, Daten3 und Daten4.

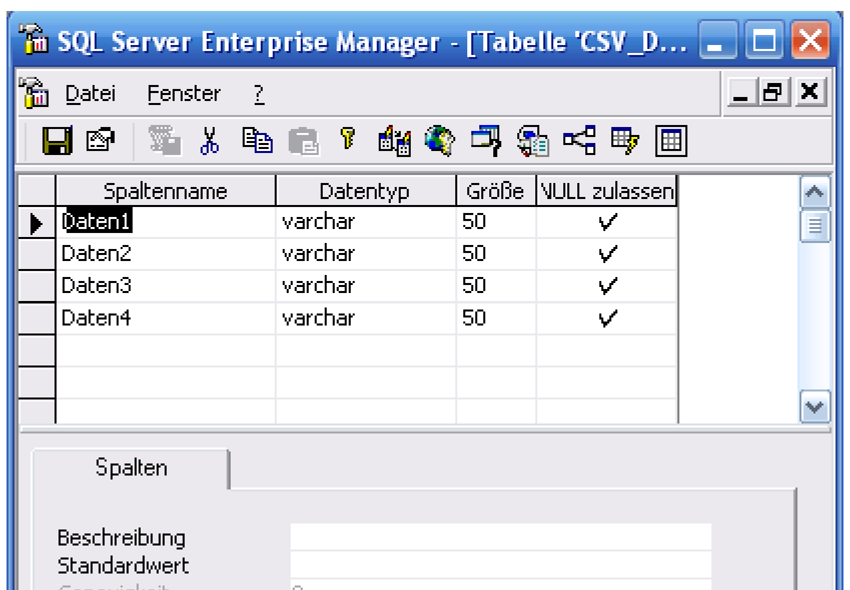


Abbildung 88: Bulk Inserts 1

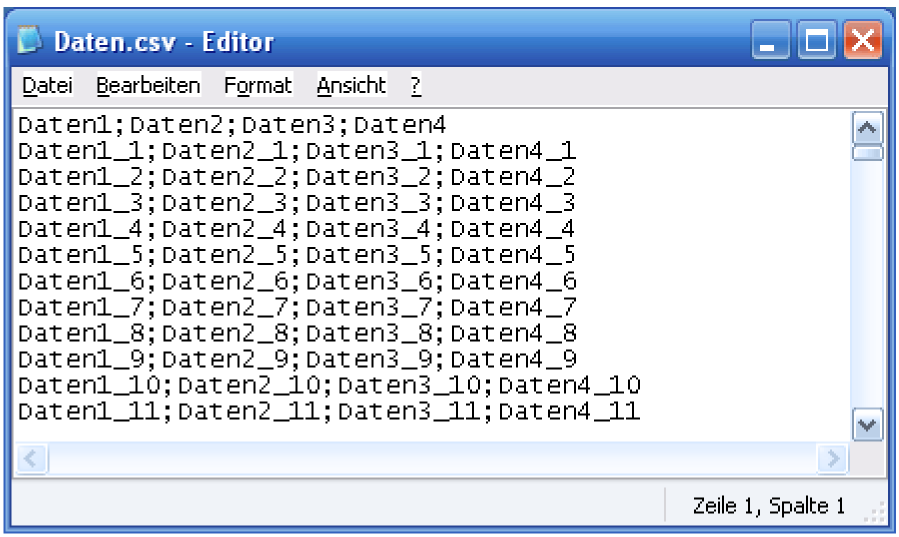


Abbildung 89: Bulk Inserts 2

Wichtig für die `BULK INSERT` Anweisung ist nun das Trennzeichen der einzelnen Daten und Zeilen, so wie die Zeilennummer der ersten Daten. Die Daten der CSV-Datei werden üblicherweise durch ein Semikolon und die Zeilen durch einen Umbruch getrennt. In der ersten Zeile sind die Spaltenüberschriften angegeben.
Die BULK INSERT Anweisung sieht dann wie folgt aus:

```
BULK INSERT <Tabelle> FROM <CSV-Datei>WITH (FIELDTERMINATOR = ';', ROWTERMINATOR =
'\r\n', TABLOCK, FIRSTROW = 2)"
```

Natürlich muss noch die Zieltabelle und der Pfad zur CSV-Datei angegeben werden. An die Parameter `FIELDTERMINATOR` und `ROWTERMINATOR` übergeben wir das Semikolon und den Zeilenbruch. Per `TABLOCK` sperren wir die Tabelle für die Zeit des Imports und mit `FIRSTROW = 2` geben wir an, dass unsere Daten ab der zweiten Zeile beginnen.
Die komplette Anweisung kann nun an ein `SqlCommand` Objekt übergeben und ausgeführt werden.

Wenn es keinen Fehler gibt, sollte das Programm unmittelbar nach dem Start bereits fertig sein.
Natürlich kann man dieses Beispiel noch beliebig ausbauen und auch die `BULK INSERT` Anweisung bietet noch wesentlich mehr Parameter an. Wie man anhand der oberen Parameter vielleicht schon erkennen konnte, kann man einen `BULK INSERT` auch auf andere Dateiformate als das CSV Format anwenden.
Das Beispielprojekt findet man unter der URL:

http://www.veloursnebel.de/Code/BulkInsert.zip

7.17. Daten-Transfer mittels SqlBulkCopy beschleunigen

Daten von einem Datenbank-System auf das andere zu verschieben ist an sich keine große Sache. Interessant wird es dann, wenn Performance gefragt ist. Wird das Ziel durch einen Microsoft SQL Server repräsentiert, gibt es seit .NET 2.0 die Klasse `SqlBulkCopy`, die hier sehr gute Dienste leistet.

Durch ein BulkCopy ist es möglich eine gesamte `DataTable` zu transferieren ohne sämtliche Commands einzeln abzusetzen.

An dieser Stelle sei darauf hingewiesen, dass ein `SqlBulkCopy` nur mit einem Microsoft SQL Server als Ziel funktioniert. Die Quelle kann ein x-beliebiges Datenbank Management System darstellen.

Die Verwendung gestaltet sich einfach:

```
public static void BulkCopy (SqlConnection connection, string
destinationTable, DataTable dataTable)
{
SqlBulkCopy sbc = new SqlBulkCopy(connection);
sbc.DestinationTableName = destinationTable;
sbc.WriteToServer(dataTable);
}
```

In diesem Beispiel wird einer Methode eine gültige Verbindung zu einem SQL Server, der Name der Zieltabelle, als auch eine `DataTable` übergeben. Mit diesen Informationen ist es möglich die gesamten Daten der `DataTable` in die Zieltabelle zu kopieren. Hier gilt es zu beachten, dass die Methode `WriteToServer` weitere Überladungen besitzt, mit der beispielsweise auch ein `DataRow`-Array kopiert werden kann.

Der Typ `SqlBulkCopy` bietet jedoch noch weitere Möglichkeiten. Durch die Eigenschaft `NotifyAfter` ist es möglich eine Benachrichtigung zu definieren. Wird beispielsweise ein Wert von 1.000 gesetzt, wird nach 1.000 verarbeiteten Datensätzen das Event `SqlRowsCopied` ausgelöst. Dadurch ist es möglich, eine Fortschrittsanzeige zu realisieren.

Weitere Informationen zum Thema Bulk-Copies mit dem Microsoft SQL Server finden sich unter http://msdn2.microsoft.com/en-us/library/system.data.sqlclient.sqlbulkcopy.aspx..

7.18.Volltext-Suche und SQL Server 2005 Express

Wer unter SQL Server 2005 Express die Volltext-Suche verwenden möchte, wird diese eventuell nicht finden können bzw. nicht aktivieren können (der Dienst MSFTESQL wird nicht installiert und ist somit auch nicht verfügbar).

Abhilfe schafft die Installation des Microsoft SQL Server 2005 Express Edition with Advanced Services Service Pack 2. Hier ist nun alles dabei was man braucht.

Nein, nicht ganz: Das Microsoft SQL Server Management Studio Express Service Pack 2 darf auch nicht fehlen.

7.19.Upgrade auf SQL Server 2005

Wer seine eigenen SQL Server 2000 auf 2005 updaten möchte, oder dies für einen Kunden tun muss, der wird sicherlich gerne auf hilfreiche Ressourcen zurückgreifen. Hier zwei Links die für Unterstützung sorgen:

SQL Server 2005 Upgrade Technical Reference Guide

360 Seiten Lesespass zum Thema Ugrade auf SQL Server 2005 in allen Formen und Varianten, mit möglichen Migrations-Pfaden und vielen weiteren Informationen.

Microsoft SQL Server 2005 Upgrade Advisor

Der Upgrade Advisor unterstützt bei den vorzunehmenden Schritten und informiert darüber. Anzumerken sei: Der Advisor unternimmt keine Änderungen an der bestehenden Konfiguration.

8. Sonstiges

8.1. Verwendete Ports von Microsoft Produkten

Das ist zwar jetzt nicht das Kerngebiet meines WeWeblogs, aber nachdem ich zufällig auf diese Übersicht gestoßen bin, möchte ich sie nicht vorenthalten, da es vielleicht für den einen oder anderen nützlich sein könnte.

Network Ports Used by Key Microsoft Server Products

8.2. IIS Fehler: Fehler beim Zugriff auf die IIS-Metabasis. Was tun?

Für diejenigen, die nachfolgenden Fehler erhalten, habe ich eine ganz simple Lösung bei der Hand:

```
Serverfehler in der Anwendung /TestIIS.
-----------------------------------------

Fehler beim Zugriff auf die IIS-Metabasis.
Beschreibung: Unbehandelte Ausnahme beim Ausführen der aktuellen Webanforderung.
Überprüfen Sie die Stapelüberwachung, um weitere Informationen über diesen Fehler
anzuzeigen und festzustellen, wo der Fehler im Code verursacht wurde.

Ausnahmedetails: System.Web.Hosting.HostingEnvironmentException: Fehler beim
Zugriff auf die IIS-Metabasis.

Das zur Ausführung von ASP.NET verwendete Prozesskonto muss über Lesezugriff auf
die IIS-Metabasis (z.B. IIS://servername/W3SVC) verfügen. Informationen zum Ändern
der Berechtigungen für die Metabasis finden Sie unter
http://support.microsoft.com/?kbid=267904.

Quellfehler:

Beim Ausführen der aktuellen Webanforderung wurde einen unbehandelte Ausnahme
generiert. Informationen über den Ursprung und die Position der Ausnahme können
mit der Ausnahmestapelüberwachung angezeigt werden.

Stapelüberwachung:

[HostingEnvironmentException: Fehler beim Zugriff auf die IIS-Metabasis.]
System.Web.Configuration.MetabaseServerConfig.MapPathCaching
(String siteID, VirtualPath path) +3492202
System.Web.Configuration.MetabaseServerConfig.
System.Web.Configuration.IConfigMapPath.MapPath
(String siteID, VirtualPath vpath) +9
System.Web.Hosting.HostingEnvironment.MapPathActual
(VirtualPath virtualPath, Boolean permitNull) +163
System.Web.CachedPathData.GetConfigPathData(String configPath) +382
System.Web.CachedPathData.GetConfigPathData(String configPath) +243
System.Web.CachedPathData.GetApplicationPathData() +68
```

```
System.Web.CachedPathData.GetVirtualPathData
(VirtualPath virtualPath, Boolean permitPathsOutsideApp) +3385711
System.Web.Configuration.RuntimeConfig.GetLKGRuntimeConfig
(VirtualPath path) +189
```

Der User mit dem dein IIS läuft hat keinen Zugriff auf die IIS-Datenbank. Entweder du stellst den User um mit dem der IIS läuft, oder du führst folgenden Befehl im Verzeichnis %Systemroot%Microsoft.NETFrameworkv2.0.50727 aus:

```
aspnet_regiis -ga <WindowsUserAccount>
```

8.3.C#: Google Web API schon getestet?

Als ersten Schritt muss man sich die Google API unter http://www.google.com/apis/ downloaden und sich einen Account erstellen. Nach dem Account bekommt man eine Google ID zugesendet, mit der 1000 Requests pro Tag durchgeführt werden können.

Nun, als nächsten Schritt einfach ein neues Projekt im Visual Studio erstellen. Nun eine Web-Referenz auf http://api.google.com/GoogleSearch.wsdl erstellen und dem Teil am besten den Namen Google geben. Nun ist das wildeste erledigt.

Eine Abfrage sieht dann in weiterer Folge so aus:

```
Google.GoogleSearchResult r = s.doGoogleSearch(googleID, keywords,
0, 10, false, "", false, "", "", "");
int estResults = r.estimatedTotalResultsCount;

Google.ResultElement[] elements = r.resultElements;
```

Damit lässt sich dann schon etwas machen. Und beispielsweise könnte eine sehr einfache Abfrage-Anwendung so aussehen:

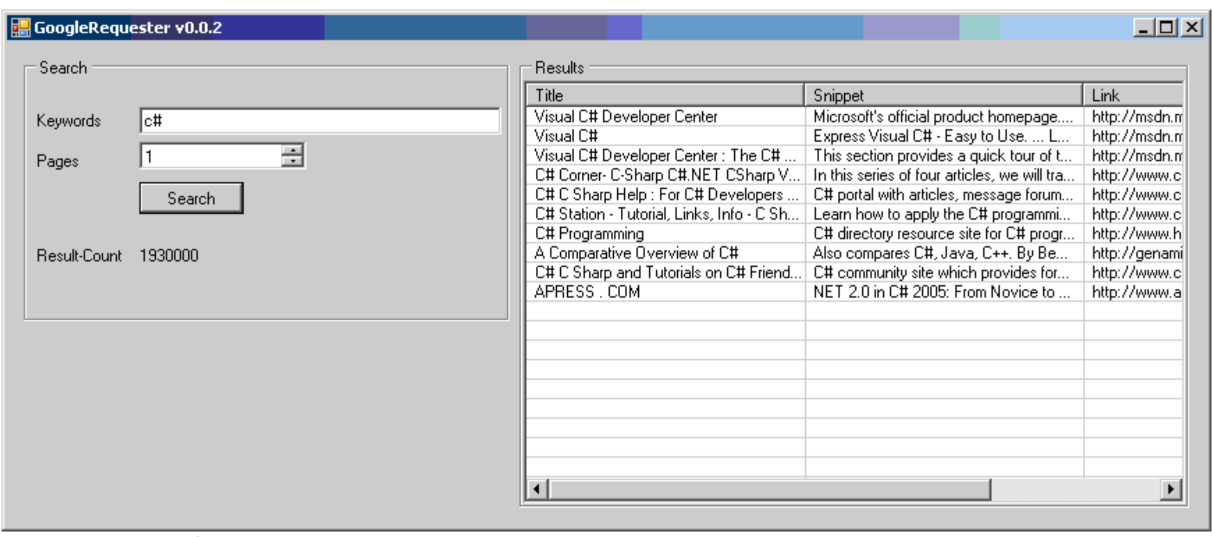


Abbildung 90: GoogleRequester

8.4.Recent Projects in VS 2005

Werden sehr viele Projekte im Visual Studio angelegt, sei dies zu Testzwecken oder anderen Gründen, erscheint die Liste der **Recent Projects** bald überfüllt. Um diese zu leeren oder nur um bestimmte Einträge zu löschen, muss dies über die Registry erledigen. Die Einträge sind im nachfolgenden Schlüssel zu finden:

```
HKEY_CURRENT_USER\Software\Microsoft\VisualStudio\8.0\ProjectMRUList
```

Überflüssige Einträgen können daraus einfach gelöscht werden. Es muss jedoch auf die korrekte Nummerierung geachtet werden (File1, File2, usw.).

8.5.IE 6.0 Kontextmenü

Es gibt Programme, die sich in das Kontextmenü des Internet Explorers eintragen. Möchte man diese Einträge entfernen, sollte man unter folgendem Pfad in der Registry schauen:

```
HKEY_CURRENT_USER -> Software -> Microsoft -> Internet Explorer -> MenuExt
```

Dort einfach die unerwünschten Einträge löschen.

9. Abbildungsverzeichnis